Deanship of Graduate Studies
Al-Quds University

XMLBB: XML Builder for Blind Programmers

Abeer Abdel-Hamid El-Haj Ali

M.Sc. Thesis

Jerusalem – Palestine

1432 / 2011

Deanship of Graduate Studies
Al-Quds University

XMLBB: XML Builder for Blind Programmers

Abeer Abdel-Hamid El-Haj Ali

M.Sc. Thesis

Jerusalem – Palestine

1432 / 2011

Deanship of Graduate Studies
Al-Quds University

XMLBB: XML Builder for Blind Programmers

Prepared By:

Abeer Abdel-Hamid El-Haj Ali

B.Sc from Birzeit University, Palestine

Supervisor:

Dr. Badie Sartawi

A thesis submitted in partial fulfillment of requirements for the degree of masters of computer science from computer science department of Al-Quds University

1432 / 2011

Deanship of Graduate Studies
Al-Quds University

Thesis Approval

XMLBB: XML Builder for Blind Programmers

Prepared by: Abeer Abdel-Hamid El-Haj Ali
Registration No: 20714038

Supervisor: Dr. Badie Sartawi

Master thesis submitted and accepted. Date: ……..……

Names and signatures of the examination committee members:

1- Head of Committee: ……..……...……. Signature: ……...……..
2- Internal Examiner: ……..……...…... Signature: …….
3- External Examiner:. ……..……. Signature: …….
4- Committee Member: ……..……..…….. Signature: …….

Jerusalem – Palestine

1432 / 2011

DEDICATION

This thesis is dedicated to my parents and my family who missed me at home while working at my thesis, but never stopped supporting me. And for sure to all blind people in this world who illuminated the way of challenges to us to follow them without fear of darkness or stumble.

Signed……

Abeer Abdel-Hamid El-Haj Ali

Date: ……

Declaration

I certify that this thesis submitted for the degree of Master is the result of my own research, except where otherwise acknowledged, and that this thesis (or any part of the same) has not been submitted for a higher degree to any other university or institution.

Signed……

Abeer Abdel-Hamid El-Haj Ali

Date: ……

Acknowledgement

First of all I want to thank Allah for giving me the strength and patience to finish this thesis.

Second, I want to express my gratitude to all the people who have given me their support, especially Dr. Badie Sartawi, and to all computer science department staff; Dr. Nidal Kafri, Dr. Raed El Zaghal, Dr. Rasheed Jayousi and Dr. Wael Hassoneh.

And Last but not least many thanks to Samia Harab who gave me a lot of her time testing and evaluating the tool

الملخص:

يعتبر تطوير لغة برمجة خاصة بالأشخاص الكفيفين أمراً محفزاً لكثير من الباحثين، فأغلبية لغات البرمجة موجهه للناس المبصرين، خاصة الجزء المرتبط ببناء واجهة النظام. حتى أن لغات البرمجة الخاصة بالكفيفين لا يمكن استخدامها بشكل احترافي، فهي غالباً ما يتم تطويرها بهدف تعليم الناس المبتدئين مهارات البرمجة دون أن تمكنهم من الوصول إلى درجة الإتقان.

الهدف الأساسي من أداة بناء لغة الترميز الموسعة هو تمكين الكفيف من تطوير وبناء التطبيقات والأنظمة بسهولة ويسر دون الحاجة للمساعدة من مبصرين، علماً بأن هذه الأداة يمكن استخدامها من قبل الأشخاص المبصرين كذلك.

في هذه الدراسة قمنا بتطوير أداه بناء لغة الترميز الموسعة والتي تمكن الكفيف من بناء أنظمة وتطبيقات متكاملة (شاشات، وتقارير وقواعد بيانات) باستخدام لغة الترميز الموسعة "XML"، هذا بالإضافة إلى أن هذه التطبيقات يمكن استخدامها من قبل أشخاص كفيفين أيضاَ.

أداة بناء لغة الترميز الموسعة هي أداة تقوم أساساً على مفهوم التحكم بواسطة اختصارات لوحة المفاتيح أو بواسطة الخطاب "Speech-Driven" وشاشات الحوار "Dialog" والتي تمكن الكفيف من تنفيذ الأوامر عن طريق الحوار المباشر مع النظام.

Abstract:

The evolution of software applications from Text mode to GUI, "point-and-click" and "drag-and-drop" interface simplifies and facilitates the usage of computer systems by sighted people. However, and unfortunately, GUI adds lots of restrictions for blind people and prevents them from using many capabilities and advantages of computer software; and they find themselves divested of using computer software and applications. In fact, this issue becomes more complicated when blind people try to use programming languages, especially when developing applications interfaces, which in turn affects negatively the numbers of blind programmers in the world.

The development of programming language for blind programmers is a motivating issue in computer science. Most programming languages focused on sighted users. Even the current blind-oriented programming languages can't be used for professional issues; they are mainly used to help novice blind programmers to start learning programming.

Our preliminary objective is to help blind people to develop their own application, but obviously, this tool can be easily used by sighted people as well.

In this research, we have developed a tool (XMLBB: XML builder for blind programmers) that enables blind programmer to build a complete web application (forms, reports & database) using XML technology.

XMLBB is a speech-driven tool that can be used by blind programmers to execute commands by speech. Using XMLBB, a blind programmer can develop a complete application that can also be used by blind users.

Table of Contents

List of Figures

List of abbreviations

Abbreviation	Full Name	Page No
WHO	World Health Organization	1
GUI	Graphical User Interface	2
XML	EXtensible Markup Language	2
XMLBB	XML Builder for Blind programmers	2
TTS	Text To Speech	3
HTML	Hyper Text Markup Language	4
XQuery	Extensible Markup Language Query	4
XFORM	Extensible Markup Language Forms	4
XHTML	Extensible Hyper Text Markup Language	4
APL	Audio Programming Language	6
DLL	Dynamic Link Library	7
W3C	World Wide Web Consortium	8
CSS	Cascading Style Sheet	10
XPath	XML Path	10
XML DOM	XML Document Object Model	10
ASP	Active Server Pages	10
PHP	Hypertext PreProcessor	10
XSD	XML Schema Definition	15
HTTP	Hyper Text Transfer Protocol	24
VAT	Value Added Tax	44
FLWOR	For, Let, Where, Order by and Return	46
SALT	Speech Application Language Tags	51
RDF	Resource Description Framework	71
OWL	Web Ontology Language	71

Chapter 1

Introduction:

"I have disability but I am not incapable" said Noor Ahmad, an eight year old child from "Helen Keller School" for blinds in Palestine. This is the worst problem faces most of blind people in the world; the lack of opportunity but not the ability. Actually, employers do not believe in blinds' capabilities and rarely even try to examine them. This attitude contributes in the increasing number of unemployed blinds in the world and consequently increases the poverty percentage within this group of people.

According to World Health Organization (WHO) statistical report in October 2009, 314 million people are visually impaired around the world; and 45 million of them are totally blind. These numbers indicates the dire need to pay more attention and solicitude for the blind people

To help blind people to incorporate and interact with community, to make them more independent and to ease their lives, a lot of facilities have been developed and enhanced in many areas. This includes adapted environment, tools, equipments, techniques, trained animals, robots, etc..., where some of are cheap while others are very expensive.

Computers technology is one of the areas that have undergone a considerable development and improvement, mainly because it is an easily-reached huge source of information and represents essential skills for many positions. These facilities vary from adapted hardware such as special keyboard, synthesizer, Braille display devices and printers, and other software aids such as screen readers, screen magnifies, voice recognition and many others. Initially these facilities were efficient enough to help blinds to use computers as easily as sighted do, which enabled several blind people to compete in many computer-related professions such as programming and system analysis.

However, the evolution of computer software from text mode to GUI interface such as icons, buttons, links and other visual elements has emerged serious barrier in computer accessibility by blind users. This issue affects negatively in the numbers of computer blind-users in the world and consequently blind programmers.

1.1 Objective:

The main objective of this research is to assist blind people to regain their rights in working in programming-related professions. Many researches pursue the rehabilitation and adaptation of blind people to recur to this field. Some have developed systems and workshops in order to help novice blind learners to enhance their capabilities in problem solving and thinking skills, and to attract them to this field of jobs, and mainly to prove their qualifications and capabilities. Others started to enhance available programming languages by enabling blinds to create forms and interfaces through a text-based form scripting language, but the development of these scripting languages does not get to the level of the development of the programming language itself. While some other researchers proceed with developing blind oriented programming languages that aim to help blind learners to construct their abilities and represent their understanding of programming logic by creating programs, but also these programming languages were very modest and primitive and not up to the evolution of other programming languages. Therefore it can't be used for professional issues.

The aim of this research is to handle the shortcoming of the previous approaches, by providing blinds with a tool that enable them to build a complete application independently and consequently improving their programming skills. We offer integrated model for our approach; the XML Builder for Blind programmers (XMLBB) tool. XMLBB is a speech-driven tool where the user can execute a task simply by pronouncing it, along with providing users with the capability of using the keyboard shortcuts, mouse and other accessibility methods to accomplish the same task. In this approach; the blind programmers can easily create, search, append, modify, truncate, traverse and validate XML documents. In addition, programmers can create and maintain complete integrated user-applications (i.e. forms, reports and database) easily and efficiently. These user-applications are enhanced by speech recognition & synthesize technology so it can be used by blind users too.

Our objective is achieved through analyzing four main programming languages based on XML technology; XML documents, XML Schema, XForms and XQuery; and then identify the structure, syntax and semantic of the essentials objects of theses languages. Each of these languages will represent a major component in XMLBB, and then we identify how to integrate between these languages, and how to generate one component from the other according to specific criteria, taking into consideration all essentials parts of the language that accomplish our goal, while giving the user the opportunity to set his preferences and modify the automatically generated components according to the application requirement. And since most of these languages are not a stand alone, we augmented our tool with other programming languages such as XHTML, and XPath that enable the user to build integrated web application easily and efficiently.

By using XMLBB, a blind programmer can develop a complete web-based application that can be accessed via most web browsers. Developing a web-based application eases the software deployment process and facilitates the distribution of new versions of this application. In addition, it enables users to access these applications anytime and anywhere.

1.2 Research Concepts:

1.2.1. Speech Recognition & Synthesizer:

Speech recognition is an active field of research from decades. In [5] "Speech recognition is the process of automatically extracting and determining linguistic information conveyed by a speech wave using computers or electronic circuits". While Speech synthesizer, is a text-to-speech (TTS) technology that converts normal language text into speech by using a database of recorded speech of language words.

In [11], the authors realize the role of speech recognition in supporting universal access for communication and learning. Speech recognition provides an active interface for a human to interact with machines more easily; such a technology has a significant enhancement for blind and disabled peoples in using computer software, where it enables them to use a lot of computer facilities saving their time and effort in communication and integration with others.

In [1], Multimodal user interface was suggested to improve the communication of the disabled people with machine, where user can use more than the traditional keyboard and mouse, such as voice, gestures and body movement, haptic interaction, facial expressions, and others. The author of this paper relies on sound recognition tools to improve accessibility for both blind and deaf people by using text-to-speech and speech-to-text services.

Our model depends on speech recognition and synthesizer technology as one of essential accessibility method, in order to facilitate blinds interaction with computers. It enables methods for converting text to speech, speech to events and in-line phonic help.

1.2.2. XML:

XML is an Extensible Markup Language designed to store, structure, describe and transport information. Also, it provides an open standard format for defining both data and metadata. It's an easily extensible; user can use his own tag, self-describing and easy readable by both human and machine. It became the most widely used format for data exchanging and integration between different databases particularly via internet. Most databases nowadays support storing, validating and retrieving XML documents; and so it can be the most appropriate database model for blind programmers to create and develop a complete application.

XML is a platform-independent programming language. It provides a file format for representing data, a schema for describing data structure, and a mechanism for extending and annotating HTML with semantic information.

The main components of our proposed model are: XML Schema, XML document, XQuery and XForms. And since XForm and XQuery are not stand-alone languages, we embed them within XHTML document. Each component has its own specific purpose, and all participates in composing the whole application. More details about these components are below:

- **XML document:** it contains the user application data. XML documents are supplied with tags to represent and recognize each field of data. It is widely used for data exchanging between heterogeneous systems, particularly in web and

distributed applications. This approach will use the XML document as a data storage, it models user application database in XMLBB

- **XML Schema:** it is the description of XML document. It is used to define the structure and contents of the XML document, and to ensure that the XML document is valid. XML schema represents the user database schema in our model.

- **XQuery:** it is the language that can be used to extract and manipulate XML document. It is equivalent to query language in conventional databases, and also allows users to construct a new XML documents. This model will use XQuery to select, order, filter and manipulate data in XML documents. By using XQuery we can return XML documents partially or completely, that can be used in forms and reports according to application requirements.

- **XForms:** XML forms are the new generation of HTML forms. It uses a more flexible, secure and platform independent way to input data from a user. It uses XML for data definition and XHTML for data display. It consists of three parts: models to describe form data, interface to identify form control and styling criteria to display form layout. In XMLBB model the end-user can use XForms to construct the user application Interface.

- **XHTML:** it is a well-formed HTML document (i.e. An XHTML document has one root element, and properly nested XHTML elements, which are always closed, and all attributes in lowercase between double quotations). It can be used to create the application web pages.

1.3 Paper Organization:

The rest of this paper is organized as follows: The 2nd chapter shows the background, the 3rd chapter illustrates our XMLBB model and the 4th chapter elaborates on the development and implementation phase. The 5th chapter explains the testing and evaluation process, while the 6th chapter suggests future work and conclusion.

Chapter 2

Background:

The evolution of software applications from text mode to GUI "point-and-click" interface simplifies and facilitates the usage of computer systems by sighted people. But, unfortunately, it added more restrictions for blind people and prevented them from using many capabilities and advantages of computer software. For those people who are totally blind, there sight inability must be replaced with other sense as a mean of input.

GUI limits the number of blind programmers in the world as in [4, 9, 10]. Since then, many researchers and organizations have started to develop or submit solutions for this group of people. One solution was the use of Braille output devices, which convert text that is displayed on the screen into Braille characters on a 'touch-pad'. The user then 'feels' what is on the screen. Another approach is through the use of text-to-speech synthesizers working together with screen reader software such as in the multimodal interface described in [1]. Such solution is useful for using available software or application, but it's not enough for blind programmer to be able to develop their own programs. Blinds still face many obstacles in working in programming-related professions.

2.1 Related Work:

When reviewing the previous literatures that discuss approaches for developing a programming language for blind people, we found several suggested solutions. Following are some of these approaches:

In [3,8], authors developed a system that can assist novice blind programmer to start learning programming. They aim to enhance problem solving and thinking skills of novice blind learners in order to increase the number of blind programmers. In [8], the authors designed a model of a programming language (APL) for blind people with audio interface; which is not an alternative to conventional programming languages. It is just a tool that aims to motivate blind learners to start programming. It is a basic programming language,

but with a significant enhancement, that implements unconventional variable of "sound" to be manipulated by blind users. However, when tested by experts and end users, it was found that APL needs more improvement and enhancement in both grammar and functionality. By using APL, blind learners can construct new skills in programming, problem solving and logic thinking.

In [3], the authors led a workshop to encourage blind students to start learning programming; "We wanted this workshop to be not only exciting but also illustrative of the problem solving and creativity that is the core of computer science". This is achieved by using a project designed to be completely accessible by blind students, named instant messaging chatbots. Creation & customizing of this project requires from students qualification in both programming skills and artificial intelligent. By the end of this workshop, students proved their abilities in programming when a suitable environment and tool are provided.

Papers [4, 9] present the feedback of the blind programming community about APL and other blind-oriented programming languages. It is believed that blind programmers do not need a separate programming language; they need to be able to use the conventional programming language. Authors in [4,9,10] find that the barrier that blind programmers faced in using conventional programming language is in the designing stage of the applications interface, while using a text files can be sufficient to write their code and create their classes, methods, function and DLL's. So they developed a scripting language that enables blind programmers to build application forms (i.e. the part that request vision) in a simple and maintainable way, and then blind programmer can use the conventional programming language such as visual basic to write application code (i.e. business logic).

In [2], authors developed a tool that enables blind people to build a web presentation by using dialogs and wizards. The user will be instructed by sound for what type of information to insert according to the template he chooses. This web presentation is also accessible by blind.

The above mentioned solutions for blind programmer were not efficient enough; according to authors in [8], APL is a basic language with limited capabilities, where a user can only define variables, input & output variables, and use condition & loop statements. It can't be used for professional purpose (i.e. it says nothing about accessing a databases, data type

casting, text manipulation, Modular programming, etc...). While other approaches that propose a special-purpose scripting language for building user interface also have limited usefulness; only a small set of controls are defined (e.g. forms, command buttons, text boxes, combo boxes, frames, and check boxes as in [9]). However, they lack rich controls such as grids, calendars, reports, menus, etc. In addition, the scripting language is not synchronized with the programming language itself. Eventually the tool developed in [2] is only concerned about specific domain; how to present data, but it says nothing about manipulating, storing and transforming data.

Nowadays creating and maintaining databases has become an important issue for most enterprises and corporations. Many organizations have vital data that need to be manipulated, secured, maintained, retained and backed up constantly. The previous discussed approaches do not have interest in this area. In our approach we associate between the application development and database creation processes, mainly in order to provide blinds with integrated tool that enable them to build complete applications.

XMLBB is a tool that is oriented for blind programmers. It enables them to build XML database to retain application data, XML schema to define database structure and constraints, build forms to insert, edit or delete data using XForms, query a collection of data using XQuery and display and view it in several format using XHTML. XMLBB is a speech-driven tool that enables blind programmers to execute commands by speech. It is intended to facilitate the process of building applications that can be used by both blind and sighted people.

2.2 Why XML?

In our approach, we choose XML format rather than any other data format, because it is more suitable for blind programmer. It provides a file format for representing data, a schema for describing data structure and constraint, and a mechanism for extending data and annotating it with semantic information. Following are more details that explain characteristics of XML, and how it can be useful for blind peoples:

- **XML is accessible (Accessibility):** XML can be used to decrease barriers of web accessibility for disabled people. By following XML accessibility guidelines provided by W3C, a developer can design a web application that is accessible by blind people smoothly and efficiently. While using HTML for example can't satisfy

our goal since it lacks the way to define data elements. In the following chapter we will discuss some recommended accessibility guidelines in details.

- **XML is Readable by both human and machine (Readability):** any sighted person who can view XML document can easily recognize what is meant by each data element. The same is valid for the computer. A readable document by machine can be converted easily into speech, and consequently can be accessed easily by blind people.

- **Data is structured (hierarchical**): XML data resides in a hierarchical traversable structure; where a user can easily reach and edit any element in XML document. Also data in the real world is usually having hierarchical characteristics, which can be easily represented by XML document; consequently user can have enough visualization about his data structure, which enables Him to build a database schema easily.

- **XML became the standard format for data exchanging (Interoperability**): when divers systems or organization needs to work together, most probably each will have different databases schema or vender and platform. The easiest way to exchange data is by converting it into XML standardized format, since XML document structures behave consistently and can be serialized and encoded. A user will be able to import and export XML data from and into heterogeneous databases.

- **XML is extensible (Extensibility):** The user can effectively create 'extendible' tag sets that can be used for multiple applications; where each tag has a unique name, defined in XML schema, a user can easily extend XML schema with additional tags according to his application requirement.

- **XML database provides both data storage and representation (Self-Existent**): in XML document, user can easily append, truncate and search data; or even view data in different format using XHTML.

- **XML support international languages (Multilingual)**: XML support multilingual documents; Language tag is used to indicate the language of text in XML document. Any XML document may contain several languages in its data content.

- **XML is open (Openness):** XML standard is completely open and free software.

Chapter 3

XMLBB Structure and Principles:

To enable blind programmers to build integrated web application and to release them from the complexities of using GUI; we try to follow the accessibility guidelines of building user interface that are recommended in many papers and sites. We enhance XMLBB interface with several accessibility methods; the user can access any field in the interface by using speech, keyboard or mouse.

Developing a web application requires the knowledge of several programming languages. In XMLBB model, four components are included to facilitate the process of creating an integrated web application based on the XML technology; the structure of the database can be defined using XML schema, the end-user data can be retained in XML documents, the application interface can be built using XHTML and XForms, and the reports can be generated by using XQuery. Actually, these languages are not enough, as the developer may define the format of his document using CSS style sheet. He needs to traverse the XML document using XPath language. In order to insert, update or delete XML nodes, he has to use XML DOM language. And he may trigger specific action using XML Events, in addition to execute client-side script using any scripting language such as JavaScript, and finally, execute server-side script using ASP or PHP code for example. Most of programmers can't memories all of these languages syntax. In our research we study the structure of a set of these languages, and determine the minimal syntax that has to be used to achieve our desired goal of building an integrated web application, and then we transform this syntax into wizards. Blind developer can use these wizards to create any piece of code, without the concern about missed comma's or incorrect number of parentheses, or any other syntax error.

Building the application interface is not an easy process; it requires sighting ability in many aspects. To facilitate this process for the developer, we enhance the XMLBB tool by

the option of automatically generating application interface according to the selected schema objects that the developer wants to include in his form, and provide the developer with the capability of modifying the generated form according to his requirement unless it contradict with the structure of database as defined in the XML schema.

Our objective of building XMLBB tool is not only to enable blind programmer to build a complete integrated web application, but also to enhance this web application with the accessibility method that enables blind users to use it easily without the requirement of a third-party tools such as reader and magnifiers. This is done through creating an accessibility package that can be attached in any web page automatically using XForms actions and client-side scripting language.

3.1 XMLBB Model:

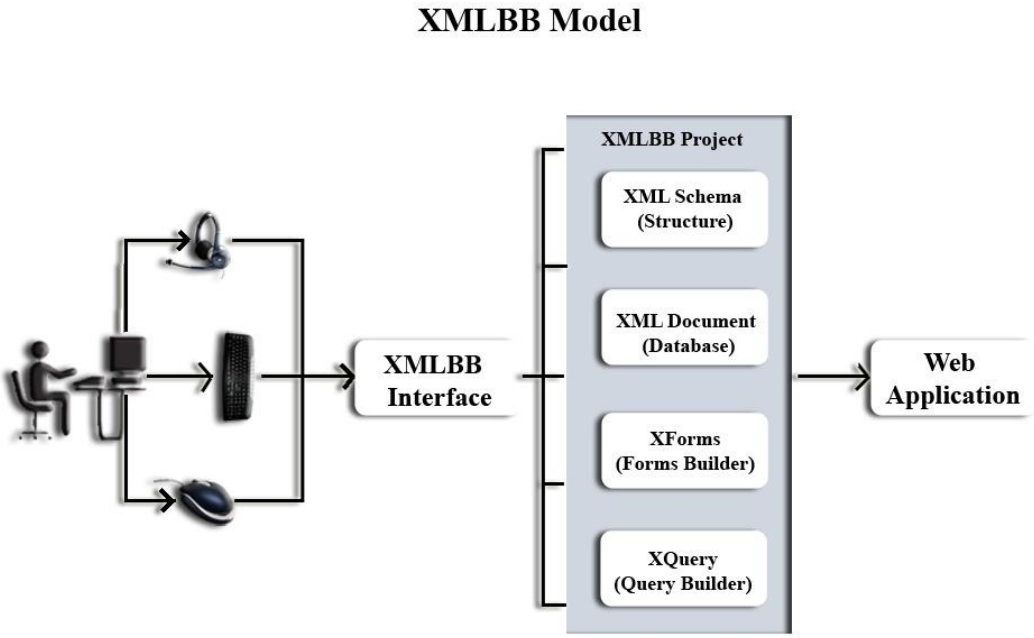

Figure (3.1): XMLBB Model

As shown in the above diagram, figure 3.1, the XMLBB tool interface can be accessed via mouse, keyboard or speech. Any user project may include several components, the XML schema which defines the XML document structure, the end-user data which is retained in XML document and based on the structure as defined in XML schema, the XForms for building the user interface, and the XQuery to view the whole or subset of end-user data in specific format for reporting issues.

Using XMLBB, the developer has the option to create, modify or delete any project. Modification of application schema objects is permitted until it may affect the already built-in forms or reports, also modification of forms is permitted if there is no contradiction with XML data, while modifying reports using XQuery is possible in any case since it is built based on XML document.

3.1.1 XMLBB Accessible Interface Design:

Many references that are found in the internet discusses the accessibility guidelines for designing a user interface in order to ensure that it can be used by a wide range of users. Most of these references recommended the separation of data from presentation using XML documents specifically in designing web applications interface. In XMLBB interface design, we try to follow-up some of these guidelines that fit our objectives.

We consolidate XMLBB interface with many features, specifically some described in paper [7], and follow-up the separation rule by applying XML concepts in the user web application. More details about each of these features are below:

- **Speech-Driven Interface:**

  A speech-driven interface is an interface that a developer speech can control the flow of the program; Actions are executed according to speech command, e.g. when a developer says "Schema", a schema interface is opened and the developer is informed by speech whether his command is executed successfully or not. A speech driven command can be enabled or disabled according to user’s request by pressing "Ctrl"+"S"

- **Keyboard Shortcut:**

  Shortcuts can be used as another mean for execution systems commands. Some people may lack the ability to pronounce some words in foreign or even mother language correctly; some prefer to use keyboard than speech, or they lack of a microphone. In such cases, the developer can press the special character "Ctrl" + some other keys to execute specific commands e.g. "Ctrl" + "P" can be used to enable or disable pronouncing any pressed character.

- **Access Key:**

  For blind person, it may be annoying if he needs to access a particular control by passing all previous controls using “TAB” key or by saying “Next” especially for crowded interface. By pressing the special character "Alt" + some specific character, this enables the user to jump to specific control on the page without using a mouse, e.g. "ALT" + "N" can be used to jump to Name text control in the interface in spite of its order or location in the interface.

- **Wizards:**

  To facilitate the application development process, wizards are intensively used in XMLBB tool. Creating, modifying or even deleting any document of user application done through a sequence of forms, where a dialogue occurs between the system and the developer is in order to help Him to insert the correct input as requested, and to enable Him to follow the process sequence correctly, and enable the developer to go back and forward within the related forms easily.

- **Pronouncing Characters:**

  For novice computer users, memorizing the location of all keyboard characters for different languages is uneasy task. At first, the user needs enough time to get used of this. By enabling character pronouncing a user can ensure that he is pressing the correct character. This facility can be enabled or disabled according to user’s request by pressing "Ctrl" + "P".

- **In-Line Help:**

  In any part of the system development, a developer can ask for help in different stages according to the active interface and control. This enables the blind developer to get enough visualization about every part of the system. The developer can get In-Line help either by saying “Help” or pressing “Ctl+H”, which provides Him with the active control description, or even he can request form level help by saying “Form Help” or pressing “F1”.

- **Text-to-Speech:**

  A sighted user can ensure the correctness of his typed data by reading it, while a system can do the same task for blind users. Also he can get the spelling of any text

character by character. The user can read any control text by saying “Read Text” or press “Ctl+T”, or “Spell Text” or press “Ctl+S” to validate his input.

- **Speech-to-Text :**

  Inserting data by using microphone is a technology that has a lot of studies and tools that aims to make this applicable. But this technology is still facing a lot of obstacles. This facility is disabled by default, but can be enabled by either pressing “Ctrl”+”T” or by saying “Enable Typing” according to the developer’s request, but it’s not recommended to be used, since it gives a law percentage of correctness and requires a training.

- **Screen Enlarge :**

  Visually impaired people may prefer enlarging text rather than using any other facility. This can help them to check any part of information they need without necessarily waiting for the system to read for them, knowing that the process of reading by eye is faster than hearing a speech. A user can enlarge his interface several times according to his request by saying “Enlarge” or pressing “Ctl+E”, or to view the interface in its maximum size by saying “Enlarge Maximum” or pressing “Ctl+’+’”.

- **Mouse :**

  Visually impaired and sighted people maybe prefer to use the mouse to access any control on the screen especially that most GUI nowadays depend on using mouse intensively, so many users are accustomed in using mouse more than keyboard. We enabled this facility to provide every user with the accessibility tool that is comfort and convenient.

- **Default Action :**

  Ordinary, a page in any application is developed mainly to execute a special action, e.g. the default action in a report is viewing it, and in a wizard is to run “next” button action. Such actions can be executed as default actions, either by pressing the "PageUp" Key or by saying "Execute”.

3.1.2 XMLBB Interface:

XMLBB interface is the main page of XMLBB tool; it includes all components required to create an integrated web project using XML technology. The user can create forms using XForms, construct database structure using XML schema, build data repository using XML document, create queries using XQuery and design reports using XHTML and styling sheet.

When XMLBB tool is initialized, the listener started automatically in order to recognize the developer speech, and the XMLBB main interface is opened with all accessibility features previously illustrated. In order to enable the blind programmer to access XMLBB tool interface, multiple alternative accessibility methods are available to a developer to execute a command. He may either click the command icon by mouse, or press keyboard shortcuts, or even pronounce the command. The same command speech or shortcut may be carried out in different ways according to the context.

3.1.3 XML Schema:

The initial phase for building the end-user web application is to define its database structure. By using XMLBB schema builder component, the developer can create the XML Schema Definition (XSD) easily. It provides a mean for defining the structure, content and semantics of a schema-valid XML documents, a developer can define his schema elements, attributes and types, and set their hierarchy, order, data type, default values, data restriction, etc …

XML schemas are extensible; where the developer can reuse his schema within another schema, or define multiple schemas for the same XML document, or one schema for multiple XML document, or even create his own data type which is derived from a built-in data type. We will use XML schema as the building block of the end-user application, after creating XML schema, the developer can generate the user interface automatically. Also, he can design the application reports and queries, based on the XML Schema and XML documents.

3.1.3.1 Creating XML Schema Flowchart Diagram:

The following diagram illustrates the flowchart diagram for creating a new schema using XMLBB tool:

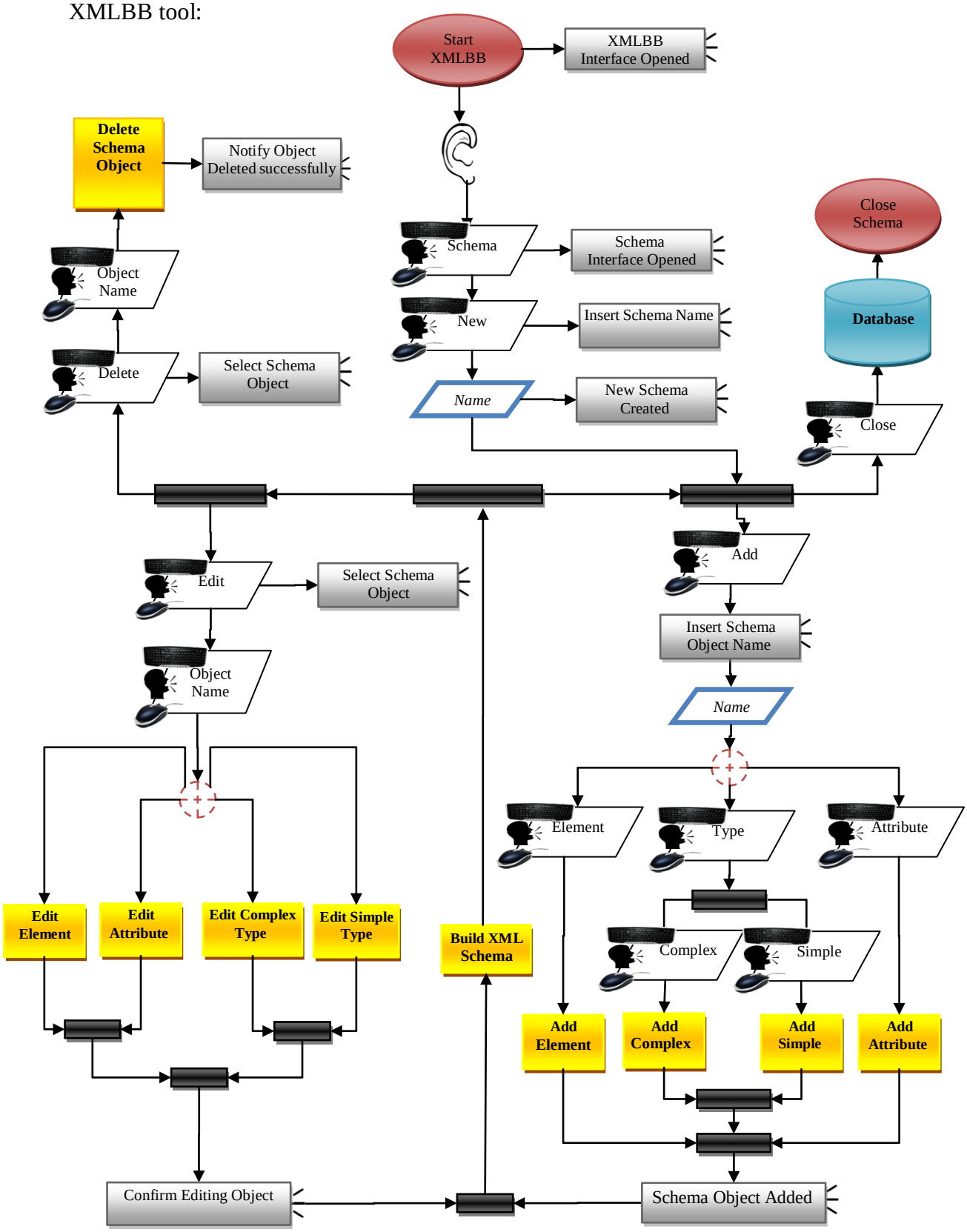

Figure (3.2): Creating Schema Flowchart

3.1.3.2 Diagram Description:

As illustrated in the above diagram, figure 3.2, when the developer runs the XMLBB tool, the speech listener starts automatically in order to recognize the developer's speech, and the XMLBB main page is opened. The developer has the option to create a new project, modify or even delete an existing one. When creating a new project, the first step is to create the database structure, which is determined by analyzing the system needs and requirements, and then define it using XML schema. The above diagram shows the stages of creating a new XML schema by using XMLBB tool. In order to open the schema interface, the developer has to say "Schema" or press "Alt+S", then he may say "Create" or Press "Alt+C" to create a new schema document, then the schema document creation wizard is initiated, and the developer is prompted to insert the schema name and path, which have to be unique in the project level. Then an empty schema is created, and the developer is prompted to create the schema objects. First he has to insert the schema object name, which has to be unique within the schema, and then he has to select the kind of the schema object he wants to create. He can either create element, attribute, simple or complex type.

According to [12,13,14], schema element defines an element of XML document that contains text and has a type of built-in, simple or complex data type; but it can't contain other elements or attributes; and its parent element is the "Schema" root element. To create a schema element, the developer may say "Element", press 'E' or select element form the list; then a new wizard will start, and the developer is prompted to insert the new element attributes. More details about creating a new schema element are illustrated in the diagram below:

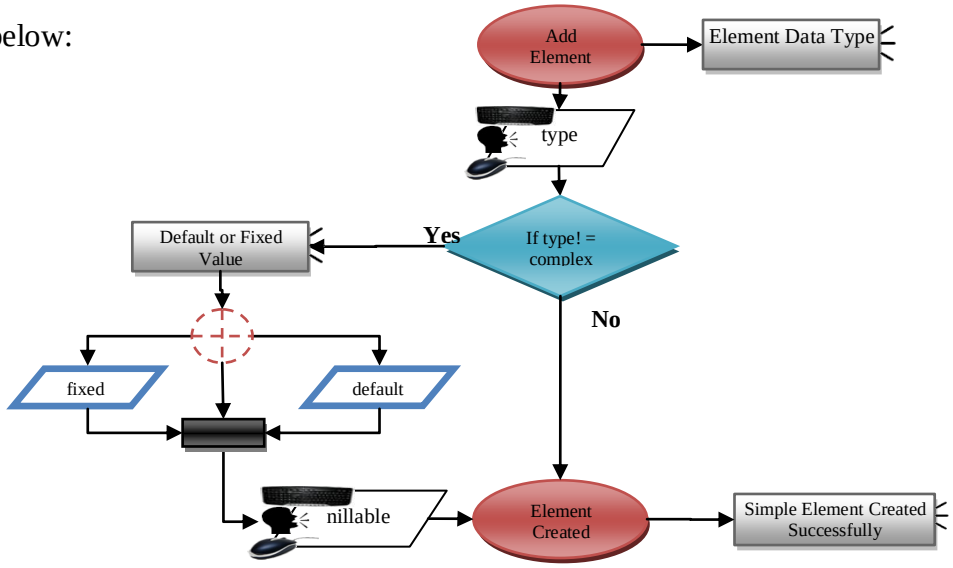

Figure (3.3): Creating Schema Element Flowchart

As illustrated in the above diagram, figure 3.3, initially the developer is prompted to choose the element data type, which may be a built-in data type or user-defined named data type. Named data type can be either simple or complex data types. If the element data type is not complex type then the developer is prompted to insert either the *default* attribute value for the element, or the *fixed* attribute value or none. Finally he has to specify whether the element value may be null or not by setting the *nillable* attribute, then he is notified whether the new schema element has been created successfully or not, and the schema definition is modified and retained.

Within the Schema root "schema" a simple attribute also can be created. But since simple attributes is not a stand-alone objects; it can't be used directly in the XML document, it has to be identified inside another element by using complex type. The following diagram shows more details about creating a new schema attribute:

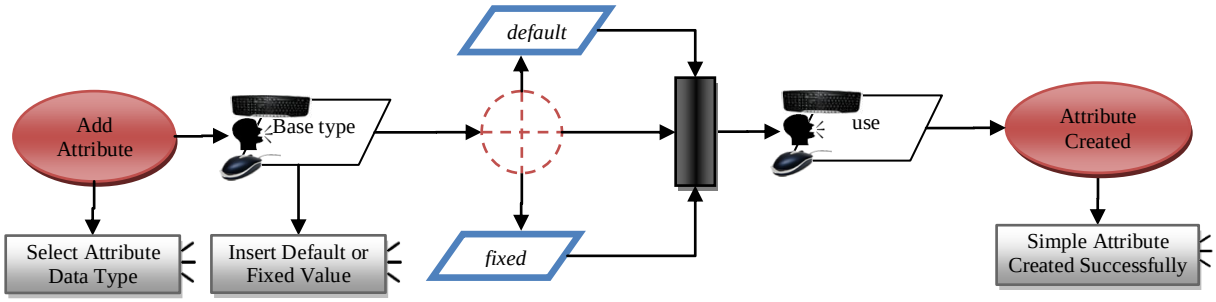

Figure (3.4): Creating Schema Attribute Flowchart

Adding a new attribute as illustrated in figure 3.4 is very similar to adding a new element in process, but here the data type can't be a complex type. It can be only simple or built-in data type. The developer has to set either a default or fixed value, or none, then he can set-in the *use* attribute, which can by either "required" to indicate that the attribute is required, or "optional" to indicate that attribute value may be left empty.

In addition to elements and attributes, the developer can also create a type object. Schema type object can be either simple or complex; a simple type can be used to define acceptable values for XML elements or attributes, where the developer can set one or more restriction on the built-in data types according to his requirement which then can be applied to either an element or attribute. While defining a complex type can be used to set the hierarchy of the XML document, and to define which element may contain other elements or attributes, it can be used to define the XML document root element, its child's elements, their order,

occurrence times, if child's elements are required or not, and if it's allowed to extend the XML document with elements or attributes that are not specified by the current schema.

The following diagram shows the process the developer can follow to create a schema simple type object:

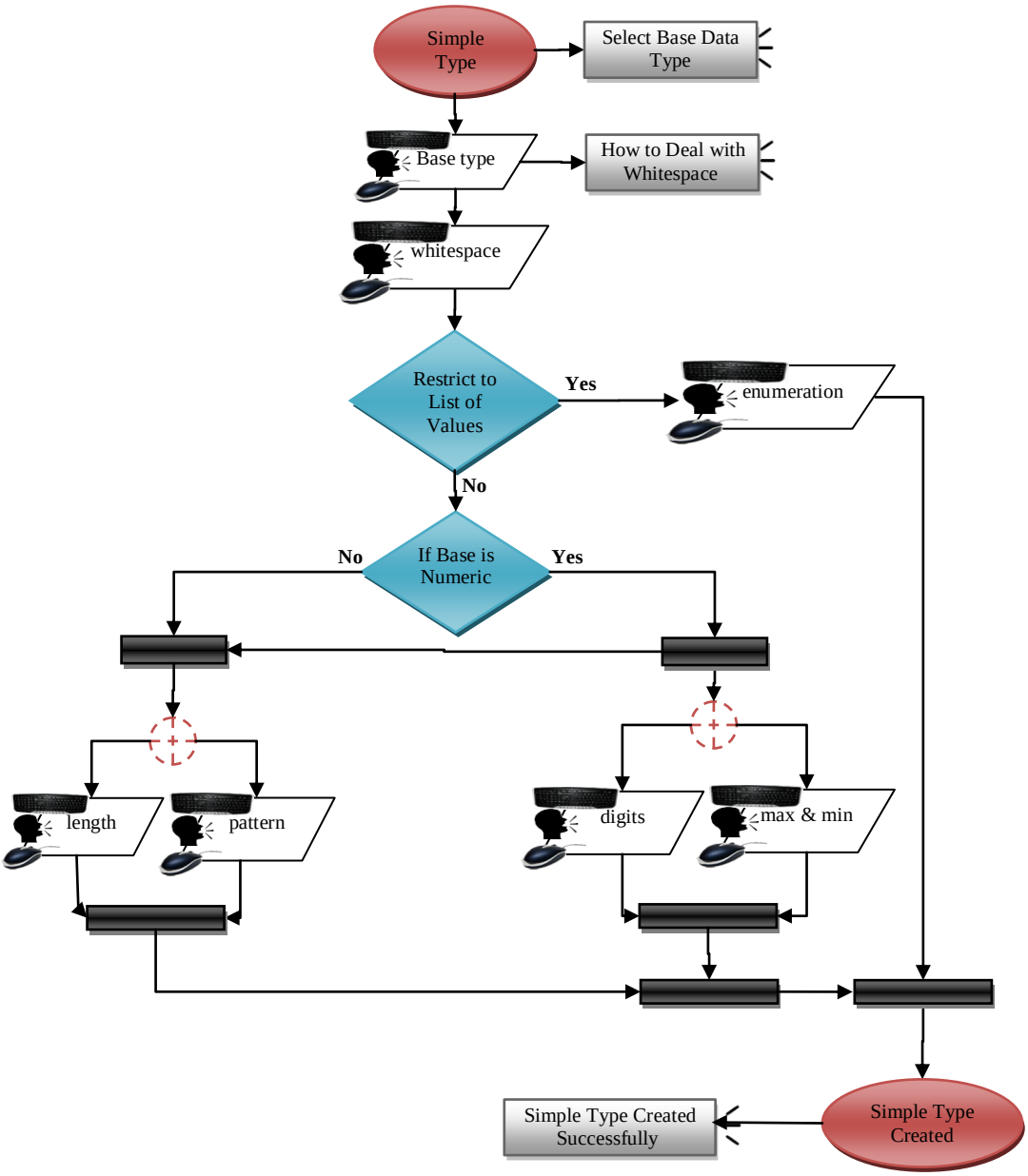

Figure (3.5): Creating Schema Simple Type Flowchart

Initially the developer has to select the base built-in data type of the simple type he wants to create, then he has to specify how can the *whitespaces* in the XML document be handled, which may take the value of "preserve" to keep all Whitespace characters, or "replace" to

replace all kinds of Whitespace characters (e.g. line feeds, tabs, spaces, and carriage returns) with spaces, or "collapse" which reduces multiple spaces into a single space. Then he may add the *enumeration* restriction, which could be used to define a list of object acceptable values, by restricting the data to a list of values, all other types of restriction will be disabled, since they become meaningless, but if the developer did not set enumeration restriction, two tracks are available to him according to the base data type selected.

If the base data type is numeric then the developer can set upper and lower bounds of the numeric value, where these values can be inclusive or exclusive that is to include or exclude the maximum value or minimum value. This is done by setting the *maxExclusive, maxInclusive, minExclusive* and *minInclusive* attributes, also the user has the option to set the number of digits allowed in float data types. He can set the *fractionDigits* value, which is the maximum number of decimal places allowed and *totalDigits* value to define the exact number of digit allowed. Whether the data type is numeric or not, more data restrictions are available; "Length" restriction can be used to set the value length, which is the exact number of characters allowed. Also the developer can set the values of *maxLength* and *minLength* attributes, to set the maximum and minimum number of characters allowed.

Finally, the developer can specify the data *pattern* to set the exact sequence of acceptable characters and patterns that are used to define a regular expression, it can be used to define character set, e.g. the developer can use square brackets "[]"to mach one character between several characters in a character set, or use hyphen"-"character inside a character set to specify one or more range of characters, while typing a caret "^" character after the opening square bracket will negate the character class, the result is that the character class will match any character that is *not* in the character class, the developer also may use character set shortcut; use "\d" for digits , "\s" for whitespace character, and "\w" for word character . if user need to repeat a character set , he can follow it by star "*" character to indicate zero or more occurrence , plus "+" to indicate one or more occurrence , or set the exact number between curly brackets"{}" , or even set the minimum and maximum occurrence number "{min,max}".also pattern attribute can be used to subtract character set from another one using hyphen character followed by the character set [class set - [subtract set]] , the subtract set itself also can be another subtracted character set , dot "."

Character can be used also to reference any single character except newline character, vertical bar "|" can be used to set alternative of list of values, round brackets "()"can be used to group a character set, which allow the user to use the repeating character for a group of characters, other pattern also are available, but we will not discuss them in this thesis.

Another object that the developer creates in the XML Schema document is the complex type, which can be used to define type which may contains either elements, attributes or both. The below diagram shows the process of creating complex type in more details:

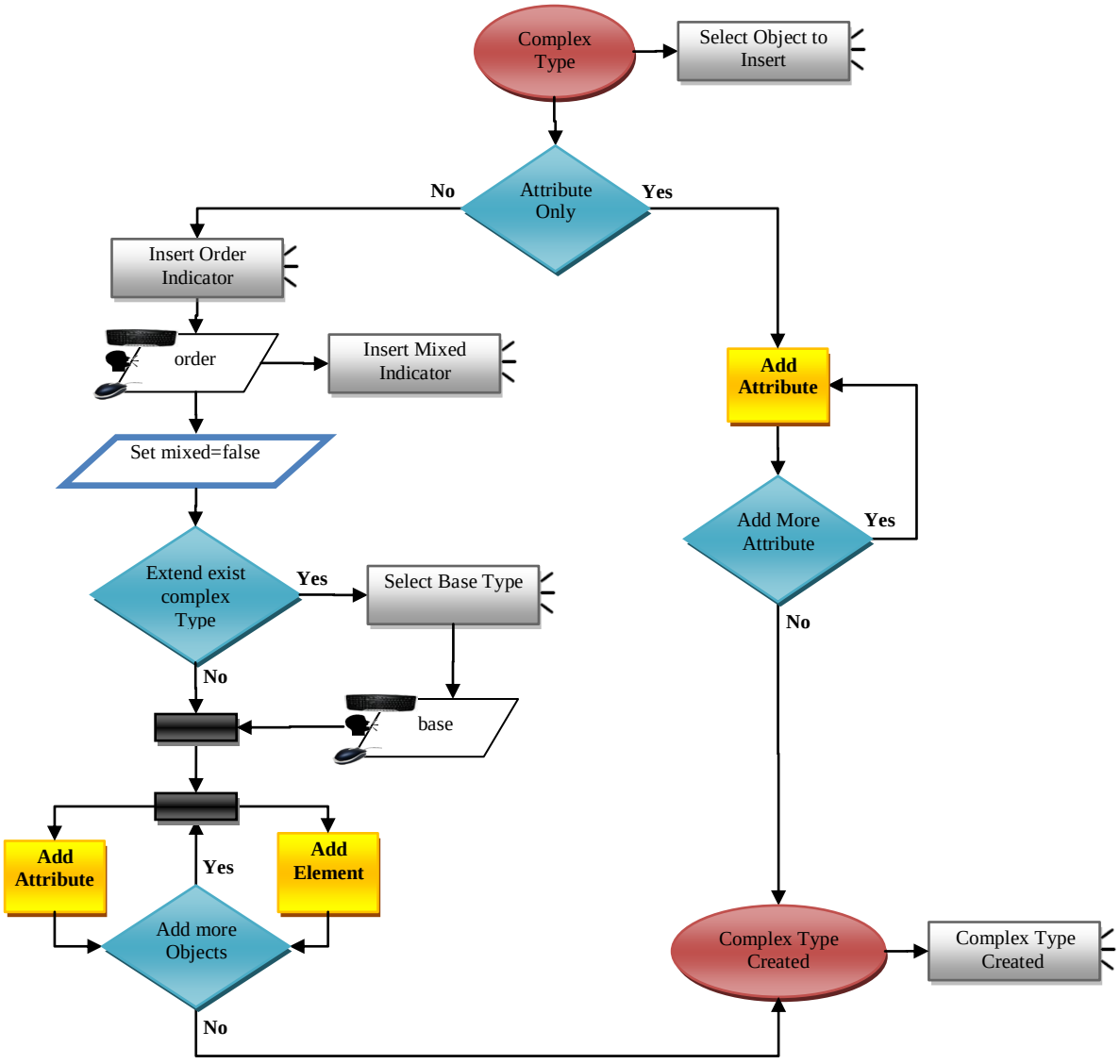

Figure (3.6): Creating Schema Complex Type Flowchart

As illustrated in the diagram above, the developer can create complex type element that consists of attributes only, in this case the developer can add as many attributes as he needs,

then the complex type will be created in the XML Schema. Adding attribute process can be done from scratch as illustrated in the diagram 3.4, or by referring to already created attribute using *ref* attribute. In this case the user has the option to reset the *use* attribute according to his requirement at this level. If the user needs to create a complex type that may contain elements and/or attributes, then the developer is prompted to insert the *order* indicator of his elements, *order* indicator may have the value of "all" to indicate that all sub elements and attributes are required, and they can occur only one time, but can appear in any order, "sequence" to indicate that the order of elements is important, or "choice" if the XML document can contain only one of these elements.

Within the order indicator element, the developer can set the *maxOccurs* and *minOccurs* attributes, which indicate that all complex type sub-elements can be repeated the same number of times. Next, the system will automatically set the *mixed* indicator attribute to "false", *mixed* indicator can be used to enable the character data to appear between the child-elements of XML document, so it is not applicable in XMLBB; since we need to refer to every part of the data in either element or attribute so that we can retrieve it easily. In addition to the indicator level, the *maxOccurs* and *minOccurrs* attributes can be set-in within the complex type sub-elements, which specify whether this sub-element can be repeated, and it's minimum and maximum occurrence, which also can be unbounded, knowing that the default value is one for both attributes.

Previous discussed criteria show how the developer can add a new object into his XML Schema. Initially, editing and deleting schema objects are not allowed, until at least one schema object is created. As illustrated in diagram 3.2, the developer can delete schema object just by selecting the abject name he wants to delete. Deleting any object requests user confirmation, if confirmed, the developer is notified if the selected object is deleted successfully or not. Deleting a schema object is not allowed if it used by any XML document, until either the XML document modified or deleted. In addition to creating, and deleting a schema object, the developer has the option to edit an existing schema object, after selecting the schema object to edit; the developer is prompted to modify each attribute according to his request. Modifying schema object also has to be valid, that is not to contradict with an existing XML document that is based on this schema object, after modifying the schema object attributes, the developer is notified that the process is completed successfully.

After creating XML Schema, the developer can build his XML Schema; that is to create XML Schema document in the user project working area as defined in the project level, a new schema is created with the default processing instruction parameters; which is intended to supply some information to the application that is processing the XML document such as version and character set encoding. Processing instructions are enclosed in a pair of "<?" and "?> ", then the processing instruction is followed by the Schema root element “<schema>” ”</schema>”, which defines the namespace parameter. Namespaces are used for providing uniquely named elements and attributes in an XML document instance, and their prefix, the default and target namespace are added to the schema from the project definition, in addition to the XML Schema namespace to indicates that the elements and data types used in the schema come from the following namespace: xmlns:xs=http://www.w3.org/2001/XMLSchema, and then set the *elementFormDefault* attribute to qualified to indicate that any elements used by the XML instance document which were declared in this schema must be namespace qualified. Then all schema objects created by the user are generated in the schema document; this includes attributes, elements, simple types and complex types.

Creating XML Schema is a very essential part of developing a web application, since it identifies the data structure of the end-user database, and consequently used to validate the XML document, in addition, project forms, queries and reports in the XMLBB tool can be generated based on the XML Schema, and consequently it requires attention and accuracy from the developer.

3.1.4 XForms:

The XForms component can be used to define the end-user application interface; it enables the developer to build a web applications interface using XHTML and XForms easily and efficiently, via speaking-dialogs or keyboard shortcuts. The end-user interface will be enhanced with features that enable a multimodal accessibility; that is, the end-user can use keyboard, mouse or even speech to access any control in the interface, and consequently it can be used easily by disabled people, and particularly by blind people.

According to [16], with XForms, the data displayed in a form are stored in an XML document, and the data submitted by the form, can be transported over the internet using

XML. Consequently, it will be more suitable to be used in our model than conventional HTML forms. In addition, XForms is a platform and device independent language; it separates data from presentation.

XForms consists of two major parts, the user interface, and the XForm model. The user interface is used to define the form controls', how they should be displayed, while the XForm model itself consists of several parts; the XForm instance which holds the skeleton of the XML document, the bind elements, which is used to define several constraints in the node-set and to bind between the model and interface, and the submission element which defines the target and method of form submission.

The XForm controls is used to set the control type, e.g. input, secrete, select, trigger …etc, and it's label, hint, alert and help message, in addition to control action, which initiates some process in a given event. For example, it shows a message to the user if his input is invalid, or insert or delete node-set from the XForm instance data. XML instance serves as place-holders of the XML structure. It may have "id" attribute for identifying the referenced model when multiple models are used in the same document, and the schema name using "schema" attribute. The bind elements can be used to set instance element or attribute data types, whether their value are optional or required, read only or updatable, and how their value may be relevant to others node-set value, in addition to the constraints that can be added to the node-set through applying simple type restrictions defined on the basis schema, or to set the calculate value of the content of the node.

The Submission element has several attributes; the form identification name, the form action which specifies the URL to where the form should be submitted, and the submitting method; which specifies how the form data are submitted to through web server. Submission method can be either get or post method. In the get method, the form data is appended to the URL in name, value pairs, so it's not suitable to pass sensitive data such as passwords or keys, while the post method sends the form data as HTTP post transaction, in addition to instance or portion of the instance to be submitted.

Several issues have to be taken into consideration when creating a web application using XForms and XHTML; the first is whether the end-user form will be used to collect, view, or manipulate the form data. When the end-user form is submitted; it may insert, update or

delete form data instance according to the context and user request, so we need a standard way to access and manipulate XML document content, in our XMLBB tool, we will use XML Dom for this purpose. The XML DOM is standard for how to get, change, add, or delete XML elements, it presents an XML document in a tree-structure format, and defines the objects and properties of all XML document elements and attributes, and how to access them. The second issue is how to bind between XForms interface controls and data instance, this is done in XForms through the bind element nodeset attribute, which is set-in according to the related nodeset using XPath language. XPath is a language for finding information in XML document, it is used to navigate through elements and attributes in an XML document. The third issue is how to respond to XForm event through actions, the XForm events track events in a form to enable the user to trigger an action in a specific event, the fourth issue that is where to set the system action, then what server-side script will be used to execute some action in the web server e.g. ASP or PHP, and consequently which web server will be used to parse and execute the server-side script such as IIS or Apache web servers, the fifth and the last issue is to automatically enhance the user interface with accessibility method that enables the blind people to use it easily without the need for a third-party software such as readers and magnifiers.

As proposed in [6]; in order to facilitate the process of creating a web form, the developer can build the end-user interface based on his XML Schema. The form controls follow the structure of the schema elements, and the user data that can be entered into the form based on the schema objects data types used. So the schema validates the user data as it's entered into the form, preventing invalid data. The developer has to set the skeleton of the XML instance, by selecting the elements to be included, then the system will generate the end-user interface automatically, while giving the developer the option to customize his interface later on to fit his application requirement; he may choose different built-in or custom styling sheet and layout or template, the developer also has the option to display the whole schema elements or a part of them. In addition, he may add some custom non-schema objects to his form such as current date and user ID or even objects from other schemas, in addition to links of others forms and reports.

The process of generating XForms based on XML Schema is discussed in details in the next section.

3.1.4.1 Creating XForms Diagram:

The following diagram illustrates the process of creating user interface using XForms component:

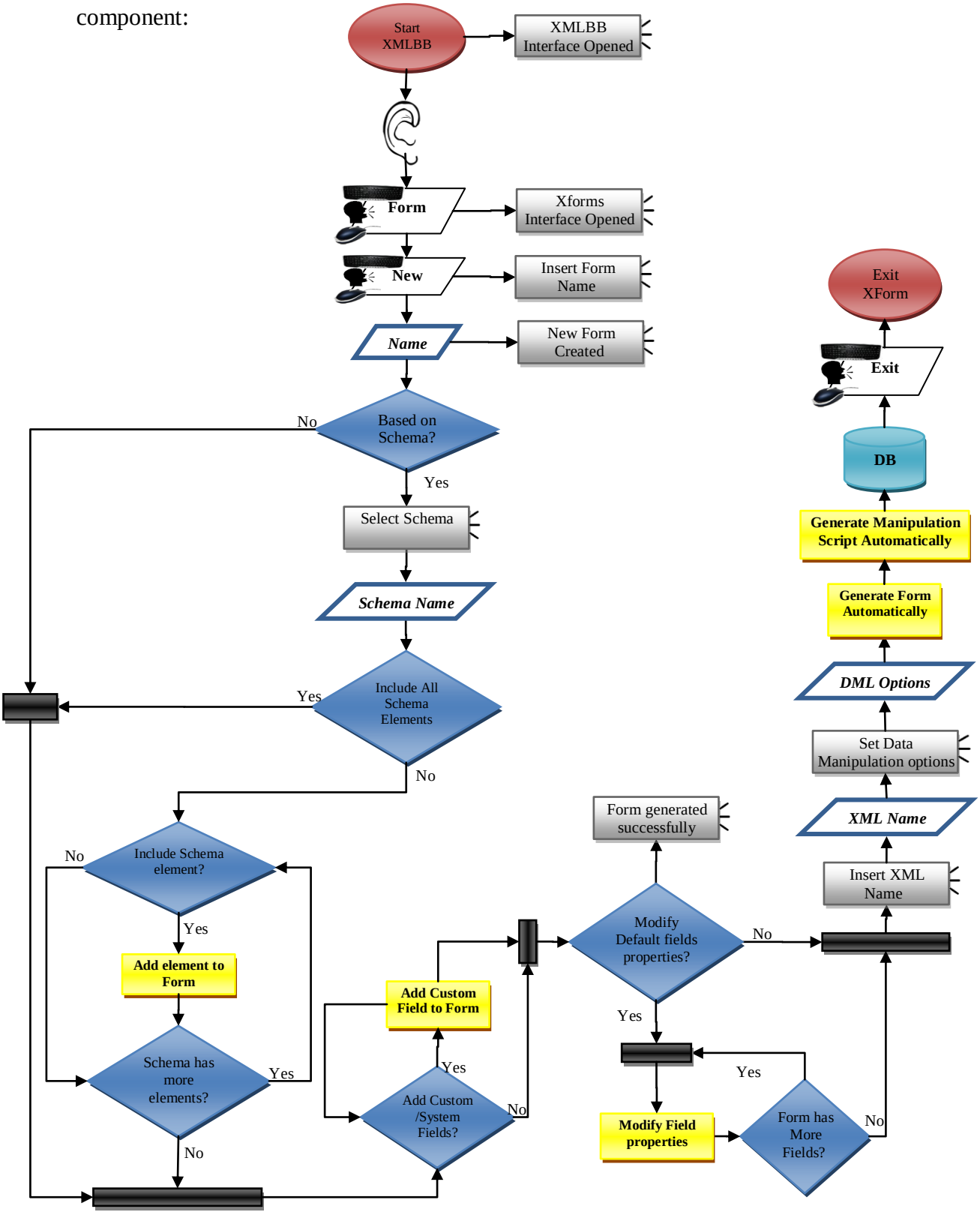

Figure (3.7): Creating XForms Flowchart

3.1.4.2 Creating XForms Diagram Description:

The above diagram show the sequence of creating new application interface; initially the developer has to pronounce “Forms” or press “Alt+F” in order to open the XForms interface, then he may say “New” or press “Alt+N” to create a new form, subsequently he is prompted to insert the form name which has to be unique in project level, in addition to the form title and description. Also, he is informed to decide whether to build his form based on an existing schema or not; there are some cases where the developer needs to create a web page that does not manipulate with XML data. For example, a main page that has links to other forms or pages in the application, in this case, he needs not to create his page based on XML schema, if the developer decides to build his form based on an existing schema document then he can specify which schema element to include in the end-user form, which can be all or subset of schema elements.

Only schema object of type element (simple or complex) can be included in the form, since attributes, simple types or complex types can’t be defined directly within the XML instance, for each element included in the XForm, a new instance will be generated. Schema instance can be either primary instance which holds the user data, or secondary which will be used as a temporary storage. For example to carry out lookup data in a list, in some cases, the developer needs to have some control such as select or select1 controls, which is filled-out by existing XML document. In this case, he can set a reference to existing XML document in his instance instead of referring to schema element.

For each object (sub-element or attribute) in the selected schema element, the developer is given the option to include or exclude it in the end-user interface, for example, he can hide some sensitive data from specific users. Whether the form is built based on existing schema or from scratch, the developer has the option to include a system and custom fields into his forms.

System fields may be current date, user name, page number, links to other pages or reports, or hyperlink to any related documents or websites, while custom field includes fields from other schemas, calculated values, aggregate functions, sequence numbers, etc…, after identifying all form field’s and elements , for each child node in the instance; sub-element or attribute, a new field will be created in the XForm, and the developer can modify any

form field presentation property according to his preferences, provided that his modification does not conflict with the based schema definition.

For each primary instance in the model, the developer has to set the XML document name that will hold the end-user data; by default it will be set-in to the instance element name, which creates a new XML document. also he may select an existing XML document, in this case, developer has to set the path of where to add the new portion of instance data, taking into consideration that the modified XML document has to be a schema-valid; the user is not allowed to append new data to XML document that affects its validation, that is ensured by comparing the schema definition with the XML document definition after appending the new instance.

For schema based XForms, the developer has to identify how XML instance data will be manipulated in his form; is it for view only, or user can insert, update or delete any portion of its content, according to his requirement, the system will automatically generate XFrom element syntax that achieve the user requirement, all operation can be done locally using XForm instance, but to apply these operation back to server, XML DOM library & server side script language will be used. And in the final step; the developer can build his form, which create a new web page into his project working area.

3.1.4.3 XForms Generation:

After specifying the form source elements' from the basis schema, the system will generate the end-user interface automatically, giving the option to the developer to update it later according to his request. Since XForms can't work alone; it have to run inside another stand-alone document; in our case we will use XHTML since our aim is to create a web application.

Some generic setting of XHTML document and XForms are generated automatically regardless of the page content; such as the XML declaration part which includes version and encoding attributes *"<?xml version="version number" encoding= "character set"?>"*, a cascading style sheet reference *"<link rel="StyleSheet" href="stylesheey name" >"*, the *<html>* tag which includes a reference to XForms, Schema, XML Events and XHTML default namespaces, and their prefixes *"<html xmlns="http://www.w3.org/1999/xhtml"*

xmlns:xf="http://www.w3.org/2002/xforms" xmlns:ev="http://www.w3.org/2001/xml-events" xmlns:xsd =http://www.w3.org/2001/ XMLSchema xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"></html>. In addition to <*head*>, <*body*> HTML tags, and <*model*> XForm tag, which includes the name of the basis schema *"<xf:model schema="#schema">"*. Many of these elements attributes' can be defined by developer in the project level, such as the document character set and default cascading style sheet.

As defined in the XForms reference by W3C in [16], ten kinds of XForms Controls are available: *input, textarea, secret, range, upload, output, select, select1, trigger* and *submit*. *Input* control can be used to input one line of text ; while *textarea* can be used to input multiple lines of text, *secrete* control can used to input passwords or any other hidden information. *range* control can be used to select a numeric value from range of values, but since in most XForms processor, range control is rendered into a control that can be accessed via mouse only; end users can't use keyboard to select a value from the range control, so we will neglect it from our process since it contrast with the accessibility method defined in our model. While *upload* control can be used to input a file name to be uploaded to server, *output* control can be used as label , *select* and *select1* controls may be used as a list of values, where *select1* allows user to choose one value from the list, while *select* allows user to select multiple values. *Trigger* control can be used to create a button that execute an action, e.g. to insert and delete elements from XML document. Finally *submit* control can be used to submit the instance data to server. XForm controls have additional support of declarative child's elements such as, *help*, *label*, *hint* and *alert* elements which are common for all controls, while *mediatype*, and *filename* can be only used for *upload* control. *Choices*, *item* and *value* elements for select and select1 controls. In our process of XForms generation we will automatically create the appropriate XForms control according to the source element included in the user form, while giving the developer the option to modify the type of the control, its child's elements, and its attribute according to the application requirement.

The data type of the source element determines the kind of the control to be generated, in addition to its value restriction, constraint, the corresponding data instance and the bind element. Since any schema element data type can be one of the following, either a built-in data type, simple type name, or complex type name. The generation of a form control of

simple instance element with built-in or simple type name data types follows the same process, while elements having a complex type name data type follow a different criterion, since they are treated as a group of elements and attributes. The following sections discuss in details the process of generating user interface from both simple and complex elements.

3.3.4.4 Generating Simple Element and Attributes:

The following is the syntax of a simple element which has either a built-in or a simple type name data type:

<element *name*="any_element" *type*="Built-in|simple type name" *default*=”Default_value” *fixed* =”true|false” *nillable*=”true|false” *minOccurs*=”000” *maxOccurs*=”999” />

For a simple type element, when generating its corresponding control, by default it will be created as *input*, *output* or *select* control, but the developer has the option to modify it later according to the context and nature of the data he needs to insert or display. If element type is built-in or the value to be inserted is not restricted to the list of values then *input* control is generated and the developers can modify it to either *textarea*, *secrete*, *output* or even *upload* control to input a file name. But, in case that the source schema element data type is simple type name, and having enumeration restriction to a list of values then *select* controls can be created as default control, and can be modified later by user to *select1* control. If the developer wants to display user-data without allowing any modification, that is when element has a *fixed* attribute is set-in to true, then *output* control will be created. *Label*, *help*, *alert* and *hint* child elements for each XForm control will set-in by default to the element name concatenated with the control type, and can be modified later by the developer to set more declarative information.

The XForms elements have some attributes that can be devised from other schema attributes; most of these attributes can be set-in using the bind element. Following is the syntax of the XForm bind element:

<bind id=”xxx” *nodeset*=” instance (‘instance-id’)/element_path” *required*=”true|false” *type*="aaa" *readonly* =”true|false” calculate =” expression” relevant=”XPath reference” constraint=”boolean function|logical condition”/>

The *nodeset* attribute is used to set the reference to the instance element using XPath, and will be set-in automatically according to the instance id and the element path. Most data type values in XML are applicable in XForms, but in XForms, other built-in and derived data types are available to the user, and can be modified by developer later on. But if the schema data type is not applicable in XForms by default a string data type will be used.

When the schema element data type is a simple type name then most restrictions defined in the schema are applied to the XForms control automatically, i.e. the developer can apply several kinds of restriction in his schema using simple type and it will be applied automatically to the XForm control when setting its data type to the restricted simple type name. For example; the *maxExclusive* and *minExclusive* attributes can be applied automatically to XForms, if defined in the simple schema type. In order to make these restrictions clear to end-user, a default hint message will be added to the form control showing the type of the restriction used, i.e. if the simple type restriction is used to set the minimum and maximum value of the element then the *hint* and *help* messages will include a notification message that this value can't be less or greater than a specific value.

readonly in XForms has the same purpose of *fixed* in Schema, while *required* attribute if set-in to true in XForms means that this control can't be empty, which can be matched by *nillable=false* in XML Schema.

The *default* schema element attribute can be set-in within the instance element 'Template' in the model part as text of the element; or element attribute value; its syntax look like the following:

<model> <instance>
<any_element any_attribute =”**default_attribute_value**”>
default_element_value
</any_element>
</instance> </model>

Other attributes are available in the bind element and can be set-in by developer later when necessary. *Calculate* attribute can be used to compute an expression such as value added tax, *relevant* attribute used to show that an instance element or attribute are related to other

one, i.e. it have no meaning if the other piece of information is not set-in, and consequently does not appear in the user form, while the *constraint* attribute can be used to add more constraint in controls. For example, it can be used to check that that the user did not enter his birth date in future.

Finally the *minOccurs* and *maxOccurs* attributes of XML schema simple element that have no directly related attribute in XForms. This is logical since this property has to be manipulated in server level not in client level, unless the developer is manipulating a local XML file. Actually, since simple elements parent is the "schema" as root element, specifying the maximum and minimum occurrence attribute is not applicable since XML document can have only one root element. The usage of these attribute is applicable only for complex elements, but we will address it in this section since complex element may consist of several simple elements or attributes, and each will follow the same simple element generation process. If control *minOccurs* and *maxOccurs* have a value to more than one, then one or more controls can be created according to access level defined in the form level; insert, delete, edit and view. If only insert operations is permitted, then one instance of the control is created. While if viewing, editing or deleting is allowed; then one column table with multiple rows will be created using the *repeat* run-time element which generate a group element implicitly that contains the run-time control, and retrieve its contents form the data instance.

On the contrast of simple elements, simple attributes can't be added directly to XForm instance, since it has to be inside other elements, but it follows a very similar process of simple element generation, so we will address it here too, while taking into consideration the few differences. The syntax of schema attribute is as following:-

<**attribute** *name*="attribute_nme" *type*="Built-in|simple type name" *fixed* ="fixed_value"
default ="Default_value" *use*="optional|required" />

Syntax of simple attribute is very similar to simple element, both have *name*, *type*, *fixed* and *default* attributes with the same purpose. While *minoccurs* and *maxoccurs* attributes are not applicable in attribute element only it can have zero or one value, and *required* attribute in simple elements is replaced by *use* attribute in simple attribute, which has the

same purpose. If *use* is set-in to optional then *required* bind attribute will be set-in to false, else it will be set-in to true.

3.1.4.5 Generating Complex Type Element:

Generating elements of complex data types requires more attention to some details. Since complex elements consist of one or more child elements and attributes, and any child element itself can be either simple or complex type element. If child element is complex then the process of generating XForm components will be executed recursively, starting from the root element, and terminating by the leaves, i.e. the inner-most sub-elements.

For each source element included in the user form, two instances are created automatically; one data instance and one template instance. in addition to one variable instance in the XForm level. The data instance holds the user data, which either instantly inserted or retrieved from user database. A template instance is created to hold the complex element structure for new records to be inserted. Both data and template instance follow naming convention in order to retrieve the name of one from the other, which is the schema source element name concatenated by "-data" or "-template" respectively. And finally variables instance called "variables" is created to hold the hidden variables in the XForm model. The variable instance will be used to hold the instance name and record index within the repeat element, and consequently refer to the correct instance when manipulating data; if multiple instances are used. also it may contain other information such as maximum and minimum occurrence of sub-elements.

The layout of the form to be generated depends on the developer preferences. For each complex element the developer is prompted to input the format of the section to be defined in the user interface; he can either present controls sections in form format, or in tabular format. In a form format, a borderless table of one column and multiple rows will be created; one row for each non-complex child element. If user wishes to view his data in tabular format, then the number of columns will be set-in according to the number of non-complex child elements and attributes. And the developer will prompted to insert the minimum number of rows to be displayed, which by default is equal to the minimum occurrence of the complex element, or he can specify the height and width of the table to be created using the tag. If number of records in user-data is more than table size,

then scrollbars are automatically created and the user can scroll up and down within the table to view all instance data. The element name will be used as a column header, and the complex element name will be set-in as table header. Whether the developer wants to present his controls in tabular or form format, he can modify the order of the controls in his interface, by setting the indexes of the controls. Note that the modifying indexes of a control will not affect the order of the elements within the XForm instance, and consequently it will be applicable for any order indicator attribute in the complex element.

Not all forms are intended be used for inserting, deleting, editing and viewing user data. When generating end user form the developer is prompted to specify what operations are permitted in his form. A menu bar will be included for each form with several triggers; new, insert, search, edit and delete, clear, next, previous, last and first, if operation is non-permitted then it will be disabled. All operations are done locally on the instance data of the XForm, and when submitting the form instance modifications are applied back to the database server.

When any section of the user-data is displayed in tabular format, all rows will be generated in run mode using *repeat* element by looping through instance data, i.e. a new row will be created for each record of the instance data, while in form format, pressing next or previous keys in the menu will display data to user according to repeat element index. In case any of the child elements has its local occurrence attribute is set-in to more than one, then the child element itself will be displayed in nested table of one column and the value of minimum occurrence rows, in this case sub-elements are viewed as bulleted list of output controls as illustrated previously in simple element generation, or in using input controls if modification is allowed. Pressing new button in menu in any table will create a new row in the table, given that the number of rows does not exceed the maximum occurrence number permitted. The maximum and minimum occurrence of the element will be validated automatically by using *while* or *if* attribute compound with the node-set *count* function to ensure that the number of elements are within the permitted range of the occurrence of the element in each table as defined in the “variables” instance.

Generating the simple element or attribute child’s of the complex elements will follow the same process of generating simple element illustrated in previous section. All sub elements

of the complex element will be generated as a group of controls; surrounded by XHTML *frame* element, and preceded by a header title.

In complex elements, the *order* and *occurrence* indicators affects the way of presenting its child controls; If *all* indicators are specified, then the complex element child's can't occur more than one time but in any order, consequently these child elements are optional, unless specified explicitly that their *minOccurs* attributes equal one, and the user has the option to rearrange the elements indexes when necessary. Since sub-element can't occur more than one time, in this case if user chooses to present his elements in tabular format only one row will be created.

If "choice" indicator is specified then only one of the child elements can occur, but all others can't. If user fills-out any of the child elements or attributes value, then all others become disabled, in this case by default we will generate a list box of all child elements names using "select1" control, and followed by one input control. The input control type and value is defined according to the selected child element to be inserted. The user has the option to modify the presentation of complex element of *choice* indicator by generating an input control to each child element in user interface, then the *readonly* attribute will be set-in automatically to true when the user edits the value of any element or attribute of the grouped controls, and consequently the user can't modify any of the others child elements.

In case of *sequence* indicator is used then the child elements have to be in the correct order and can occur zero or more times, whether the occurrence indicator is defined on the complex element level or not, then all child elements are treated as group, unless any has a local occurrence indicator.

If any child of the complex element is complex, then it will follow the same process of generating complex elements, but in separated nested section. All sections data are related; i.e. the data in each section concerned with information linked to its parent section. The form operations defined in the menu bar are also applied to all sections; insertion, deletion or edition can be done in each section. For example, the user has the option to search for specific data in any section, and the retrieved data in all section will be under the same node-set of the XML document, e.g. searching for an author of any name will retrieve all books written by the same author.

After generating the XForm components; the bind elements , the XForm instance, and the XForm actions, the end-user interface becomes ready to be used, and the developer has the option to modify some of its component attributes and elements, or even add new one .e.g. section format; tabular or form, the format of elements having choice order indicator, the format of child-elements having a local occurrence attribute, the type of control to be created, the styling attributes such the color, size and font of the control , other bind and action attribute such as *calculate, constraint , relevant , and help, hint, label and alert* sub-elements in addition to other notification and error messages, and any default settings of form generation provided that it will not cause any conflict with the schema definition.

3.1.4.6 Pseudo Code of XForms Generation:

The following pseudo code illustrates the process of generating XForms components based on schema elements in more details:

```
//procedure generate XFORM
Procedure Generate_Xform(Insert as Boolean, Delete as Boolean, Edit as Boolean, View as Boolean)
Set Global_Allow_Insert= Insert
Set Global_Allow_Delete= Delete
Set Global_Allow_Edit= Edit
Set Global_Allow_View=View
Input Edit_Default_Settings in {true|false}
If (Edit_Default_Settings) then  //if the user want to edit default settings of Xform elements
        For each child_element in instance
                Modify default_attribute(input new_attribute, attribute_name)
        End For
End if
For Each Xform_Instance In XForm //loop within instances in the Xform
If (Xform_Instance is template) then //only template instance used to generate the interface?
        Read instance_id
                If (Instance_Root_Element(instance_id) is simple_element) then
                // simple element or attribute generation
        Generate_Simple (Instance_Root_Element(instance_id),true)
        Else  // complex element generation
                Include menu-bar (new, insert, delete, edit, search, clear, next, previous, last, first)
                Disable non-permitted-operation (menu-bar, Global_Allow_Insert,
                Global_Allow_Delete, Global_Allow_Edit)
                Set parent_element= Root_Element(instance_id)
                Generate_Complex (parent_element)
        End If //Instance_Root_Element
End If //Xform_Instance
End For
End //procedure
```

Figure (3.8): XForms Generation Pseudo code

```
//generate_simple procedure

Procedure Generate_Simple(Simple as element| attribute, add_bind_element as boolean )
Begin
    Read_schema_attribute maxOccurs, minOccurs ,type ,name
    If (maxOccurs is null) then maxOccurs=1 // if user did not set a value then the default is one
    If (minOccurs is null) then minOccurs=1
    If (maxoccurs> 1) or ((minoccurs> 1)) then // control can be repeated several times
        Create_table(id= simple.name)  in XForm_UI
        If (Global_allow_insert) then //if insert is allowed
            Add_row_table(id=simple.name)
            Add_Column_table(simple.name, header=  simple.name)
            create_simple_control (Global_allow_insert,  Global_allow_edit,
            attribute_value) //built-in or simple type value
        End If // Global_allow_insert
        If (Global_allow_edit) or (Global_allow_delete)  or  (Global_allow_view)  then
            Add XForm Repeat_Element //run-time element
                Add_row_table(simple.name)
            Add_Column_table(simple.name, header=  simple.name)
            Create_simple_control(Global_allow_insert, Global_allow_edit,
        attribute_value)
            //for each element in the XForm that have maxOccurs or minOccurs > 1 we need to add
        setvalue action to set the id and index of the repeated element and instancein variable instance
            Add_action(setvalue, Instance_Name_variable,simple_name)
            Add_action(setvalue,Repeat_index_variable, null)
        End if
    Else //control needs not to be repeated
        Create_simple_control(Global_allow_insert, Global_allow_edit, attribute_value)
    End if
    Set control.attribute("label") =attribute_value
    Set control.attribute ("bind") = simple.name + "bind"
    If(add_bind_element) then   // if the control already have a bind element, do not create bind
        Create _bind(new bind_element)
        Set bind_element.attriubute ("id") = simple.name + "bind"
        Set bind_element.attriubute ("nodeset") =xpath(simple)// the element path instance
```

```
For Each defined-attribute in the simple element
Select attribute_name
        Case =”type”
                Set bind_element.attriubute (“type”) = attribute_value
                If (attribute_value is Built_In) then
                        Add control.sub_element (”hint”) =name
                        Add control.sub_element (”alert”) = name
                        Add control sub_element (”help”) =name
                Else if (attribute_value is Simple_Type) then
                        Add control.sub_element (“hint”) =
                        declarative_message(restriction_type+name)
                        Add control.sub_element (”alert”) =
                        declarative_message(restriction_type+name)
                        Add control.sub_element (”help”) =
                        declarative_message(restriction_type+name)
                End if  //type value
        Case =”fixed”
                Set bind_element.attriubute (“readonly”) = attribute_value
        Case =”nillable”
                Set bind_element.attriubute (“required”) = not(attribute_value)//opposite
        Case =”use”
                If attribute_value =”required” then
                        Set bind_element attriubute (“required”) = true
                 Else // optional or prohibited
                        Set bind_element attriubute (“required”) = false
                 End if
        Case =”default”
                Set element_value in the XForm instance (instance_id) to attribute_value
        Case Else
             Do_nothing
        End Select
End if // add bind
End For
End //procedure
```

Figure (3.9): Simple Type XForms Generation Pseudo code

```
//generate_complex
Procedure Generate_Complex (complex as complex_element) // recursive process
Begin
Set simple_child_count= 0 // number of child elements that are simple
Create_frame ("title") = complex.name
Create_table ("border") =0
Input section_format in {form|tabular} // input from user
Read order_indicator
For each child in element.childs (sorted-by-simple-elements-first)
    If (child is complex) then // complex child
        Generate_complex(child) // for complex child call the procedure recursively
    Else // simple child
    Set simple_child_count=0
    simple_child_count= simple_child_count= +1
    If (order_indicator="all") or (order_indicator="sequance") then
        If (section_format is form) then // one column table, header and control
            Add_row_table(complex.name)
            Add_column_table (complex.name , Generate_simple (child,true))
            //generate simple & bind element
        Else //one row consist of multiple columns,
            If (simple_child_count =1) Add_row_table(complex.name)
            Add_column_table(complex.name, header=child.name)
            Generate_simple (child,true) //generate simple & bind element
        End if //section format
    Else // choice indicator {only one of the child's are possible to be inserted}
    If (simple_child_count =1) then // just for the first child element ad select1 control else add items
        Add XForm Control (of type="select1", label=complex.name)
        Add XForm Control (of type="input", label=complex.name)
    End if //simple_child
    Add XForm_Control sub_element("item",label=child.name,value=child.value)
    End if //order indicator
```

```
If (Global_allow_Edit)or (Global_allow_Delete) then
        Add XForm Repeat_Element // according to number of records in nodeset( run-time mode)
        Add_row_table(complex.name)
        Add_Column_table(complex.name,header=child.name)
        Add_action(setvalue, instance_name_variable)
        Add_action(setvalue, repeat_index)
        If (section_format is tabular) then
                Generate_simple (child,false)
        End if//section format is tabular
End if//global_allow_edit
End if //child is complex
End for
End //procedure
```

Figure (3.10): Complex Type XForms Generation Pseudo code

```
// create_simple_control

Procedure create_simple_control (allow_insert as Boolean , allow_edit as Boolean , type  as
built_in|simple)

Begin
If (Allow_Edit) or (Allow_Insert) then // insert or edit allowed
        If type is Built_In then
                Add XForm Control ("kind"="input")
        Else if type is Simple_Type
                If (Simple_type_restriction is Enumeration) then
                        Add Control ("kind") ="select"
                End If
        End If
Else //view or delete only allowed
        Add XForm Control ("kind") ="output"
End If
//set bind attribute to bind element id
Set control.attribute ("node") = simple.name + "bind"
End
```

Figure (3.11): XForms Control Creation Pseudo code

3.1.4.7 Xform Submission:

```
<submission method=”get|post|delete|put” Resource|Action="URI" ref =”Xpath”
        encoding=”Character-set” Validate =”true|false” />
    <message event="xforms-event">
        Message-to-be-displayed
    </message>
</submission>
```

As shown above, the submission element within the XForm model can be used to set a collection of attributes that guide how to deal with the data to be submitted through the web server; whether to validate it first using *validate* attribute, how to transport the data using method attribute, and what to do with the returned results. Initially the developer has to specify the destination URI for submitting instance data using *resource* or *action* attribute, which can be either a web page *http://www.any_site/any_webpage*, email address using *mailto:any_mail@any_site.com?subject=any_subject or* a local file *file:file_name& path*, set the *ref* attribute to submit a portion of the instance data using XPath, then the user may specify whether and how to serialize XML data when submitting a form. It can be set-in to none, if developer is not interested in serializing the form data, also the developer can specify which protocol to use for transmitting data, this can be defined by setting the *method* attribute to either *post, get, delete* or *put*, post and put serialize instance data into xml , while get and delete method deliver data as a part of the requested URI. Several events are related to XForms submission and can be defined in this level; *xforms-submit*, *xforms-submit-done* and *xforms-submit-error* events. By default *xforms-submit-error is* set-in to show error message that notify the user if the form is not submitted successfully.

3.1.4.8 XForms Actions and Events:

In XForms, several common actions can be invoked in response to event; *action, dispatch, rebuild, recalculate, revalidate, refresh, setfocus, setindex, load, setvalue, send, reset, insert, delete, toggle* and *message*. Each action has a different purpose, such as submitting the form, and inserting or deleting records from form instance, *action* action can be used to group multiple actions within one event, *dispatch* action can be used to dispatch some form event such as submitting or resetting the form, *rebuild, recalculate, revalidate* and *refresh* actions are run by XForm processor automatically when required, rebuild causes the form

to rebuild its instance data, *recalculate* action used to recalculate instance data that has a calculate attribute, *revalidate* action causes the form to check the validity of instance data, *refresh* action causes the user interface controls to reflect the instance data, *setfocus* action is used to jump to specific control in the form, while *setindex* can be used to jump to specific row in repeating element, *load* action can be used to load a link to external resource, *setvalue* action can be used to set the value of an instance node, *send* action initiates a form submission, *reset* action resets the form to its initial values when form opened, *insert* action can be used to insert a new record to a table , while *delete* action can be used to delete one or more record from table, *toggle* action can be used to select one possible action from several cases according to specific condition ,and *message* action can be used to display a message to the user.

The XML Events provides a way to execute actions that should be performed under certain conditions, such as a button click or form submission. When these events occur, they are registered by the XForms processor. According to [16], XForms Events are classified into four categories according to its occurrences condition; initialization, interaction, notification and error indication.

Initialization events are dispatched to each model when initialized by the XForm processor. Four type of initialization events are dispatched in model level; *xforms-model-construct*, *xforms-model-construct-done*, *xforms-ready* and *xforms-model-destruct.* Which notify the user if his form is *constructed* and ready to be used.

Interaction events happen according to user interaction with the form, they can be used to notify the user that an action happened, *xforms-rebuild*, *xforms-recalculate*, *xforms-revalidate*, *xforms-refresh*, *xforms-reset and xforms-focus* events are dispatched in response to rebuild, recalculate, revalidate, refresh, reset in model level and setfocus action in control level. *Xforms-previous* and *xforms-next* dispatched in repose to user navigation in form controls forward or backward. *xforms-help* and *xforms-hint* are dispatched in response of user request for help or hint by pressing f1 or setting the focus in the control. *xforms-submit* and *xforms-submit-serialize* dispatched when pressing submit button, and starting submission serialization of XML data.

Notification events notify the user that something happened, *xforms-insert* and *xforms-delete* events are dispatched in form instance level according to successful insertion or deletion of a record in XForm instance data, *xforms-value-changed* event is dispatched in control level according to modification of control value , *xforms-valid* and *xforms-invalid* are dispatched in control level if the control value becomes valid or invalid, *xforms-readonly*, *xforms-readwrite*, *xforms-required*, *xforms-optional*, *xforms-enabled* and *xforms-disabled* control level events are dispatched when instance data attributes are changed and become readonly , readwrite, required, optional enabled or disabled , *DOMActivate* is dispatched in control level when the default action of the control is executed , such as pressing a button for trigger control, *DOMFocusIn*, *DOMFocusOut* dispatched when a form control receive or lose the focus, *xforms-select*, *xforms*-deselect used when selecting or deselecting items in select and select1 controls, *xforms-scroll-first* and *xforms-scroll-last* events are dispatched when *setindex* action are set-in to an index outside the repeat control index and *xforms-submit-done* dispatched on successful completion of submit process.

Error indication events are dispatched when error condition happens in the XForm processor; *xforms-binding-exception* occurs when error happens when processor fails to bind form element such as bind, submission and model elements referencing invalid id's, *xforms-compute-exception* happens when an error occurs when evaluating xpath expression, *xforms-version-exception* dispatched when a version check failed , *xforms-link-exception* happens when a failure on traversing a link occurs, *xforms-output-error* happens if output element fails to render its content and finally *xforms-submit-error* is dispatched according to any failure causes the form not to submit successfully.

When generating the XForm, several actions will be generated automatically in specific events in order to inform user with every important action happened in the form. A *message* action will be triggered in *Xform-ready* event in order to notify the user that the opened form is initialized and ready to be used, and as a *setfocus* action also will be created to set the cursor focus in the first control of the user interface. For each control in the form, a *message* action will be created to notify the user of the name of the control and will be displayed in *DOMFocusIn* event, and another *message* action will be added to "required" controls in the *DOMFocusOut* event to notify the user that the control value can't be empty. For each control that has a constraint on the data to be inserted, and have to be

validated according to these constraints, a *message* action will be added in the *xform-invalid* event to notify the user of invalid data is entered, a request for confirmation *message* also will be added automatically in *xform-delete* event to warn the user before doing deletion, At form submission, a *message* action will be added to notify the user when *xforms-submit-error* event occurs.

For each element in the XForm, the developer has the option to add more actions in any event, or even to edit automatically created actions. For example, the developer can add a new trigger control that when clicked calculates the VAT amount in the invoice, or to edit the message shown to the user when invalid data is entered.

3.1.4.9 Manipulating User Data:

Manipulating user-data is an essential process in any form; it can be used to insert, edit or delete user-data. Within the form menu bar, three triggers are generated automatically to facilitate this process, while giving the developer the option to modify it later according to his requirement.

If permitted as configured by developer, insert, delete and edit actions are generated automatically. As illustrated previously in the form generation, for every instance *data-instance* in the XForm model, a template instance named *template_instance* is created to hold the instance structure and default values. This template instance is used to create form controls that can be used to insert user-data, while the *data-instance* used to holds, view, edit or delete user data. To enable data insertion, a *trigger* element is created within the menu-bar, executed in *DOMActivate* event, and an *action* element is created as sub element of it, as a sub-element of the action trigger, the *insert* action will be created, with *nodeset* refer to instance data, that is where to insert the new data, and *origin* refers to template; which specifies from where to get the new data, followed by *message* sub-element to notify the user whether his record has been inserted successfully or not and a *setvalue* sub-element to clear user entered data, and reset form back to its initial state.

Editing deleting user-data triggers are also created automatically within menu-bar, using the user-data instance. To generate a process for deleting a nodeset from data instance, a trigger is created, lunched in *DOMActivate* event, with delete label, and *action* sub-element

is created, and a *delete* sub-element also created inside *action* element, followed a gain by confirmation message, and whether the deletion done successfully message or not.

Finally editing a user-data trigger is actually an insertion of new record and deletion of the old one, and consequently it is done in a process similar to the deletion process, but a two action element is created, one to insert the new data, and the other to delete old data.

Note that the whole operation is done in client memory. Unless submitting the form, all manipulation of user data will be done locally only, and the end-user will lose all modification if he closes the form before submitting.

The following pseudo code, illustrates the process of creating insert trigger in details:

```
// Create_Insert_Trigger
Procedure Create_Insert_Trigger(User_Form as Xform)
For each instance in User_Form model
    If (Global_allow_insert) then
        Add XForm Control (type="trigger", id="insert")
        Add sub_element control (parent="insert", type="label" , value="insert")
        Add sub_element control (parent="insert", id="insert_action", type="action" ,
        event=" DOMActivate")
        // note that child element is the first single sub-element of the instance root
        Add sub_element action (parent="insert_action", type="insert",  nodeset=
        "instance ('instance') /child", origin="instance('template_instance')/child")
        Add sub_element action ("insert_action", type="message", value= "ok",
        level="modal")
        For each sub_element of child in template_instance
        If default(sub_element)<>"null"
            Add sub_element action ("insert_action", type="setvalue", value= "")
        Else
            Add sub_element action ("insert_action", type="setvalue", value=
        default (sub_element))
        End if ; End for
    End If ;End for
End //procedure
```

Figure (3.12): XForms Trigger Generation Pseudo code

3.1.5 XML Document:

The user data is retained in XML document; it models the database for the user application. Actually XMLBB user needs not to create an XML document. When creating the user form the system will automatically create or modify the structure of XML document. The user has the option to retain multiple form data in the same XML document; every form can be used to insert different portions of XML document data, for example a form can be used to insert company staff details, and another one can be used to insert new babies or dependent, health insurance details, salary and allowance and many others. Since the structure of the XML document is built on one or more basis forms, a complete structure of XML document is actually built from several schema objects which are retained in the XMLBB database, so this document structure will be used to build the XQuery and XHTML documents.

3.1.6 XQuery:

According to [15], XQuery is the language used to retrieve XML document data, it can be used to generate XHTML document and summary reports, or to search for specific elements and attributes from the XML document that satisfies some conditions, and follows a defined order. XQuery language is based on FLWOR expression, FLOWR is an acronym for "for, let, where, order by and return". The "for" clause can be used to set a variable, the "let" clause is used to declare a variable and set its value, the "where" clause defines the criteria used when selecting elements, the "order by" clause defines the sort-order of the result list and the "return" clause specifies the element to be returned. The user can build the XQuery based on the XML document structure specified using XForms component. Also he has the option to retrieve data joined from multiple XML documents and view the results in either XHTML or XML document. The following diagram illustrates the process of creating XQuery:

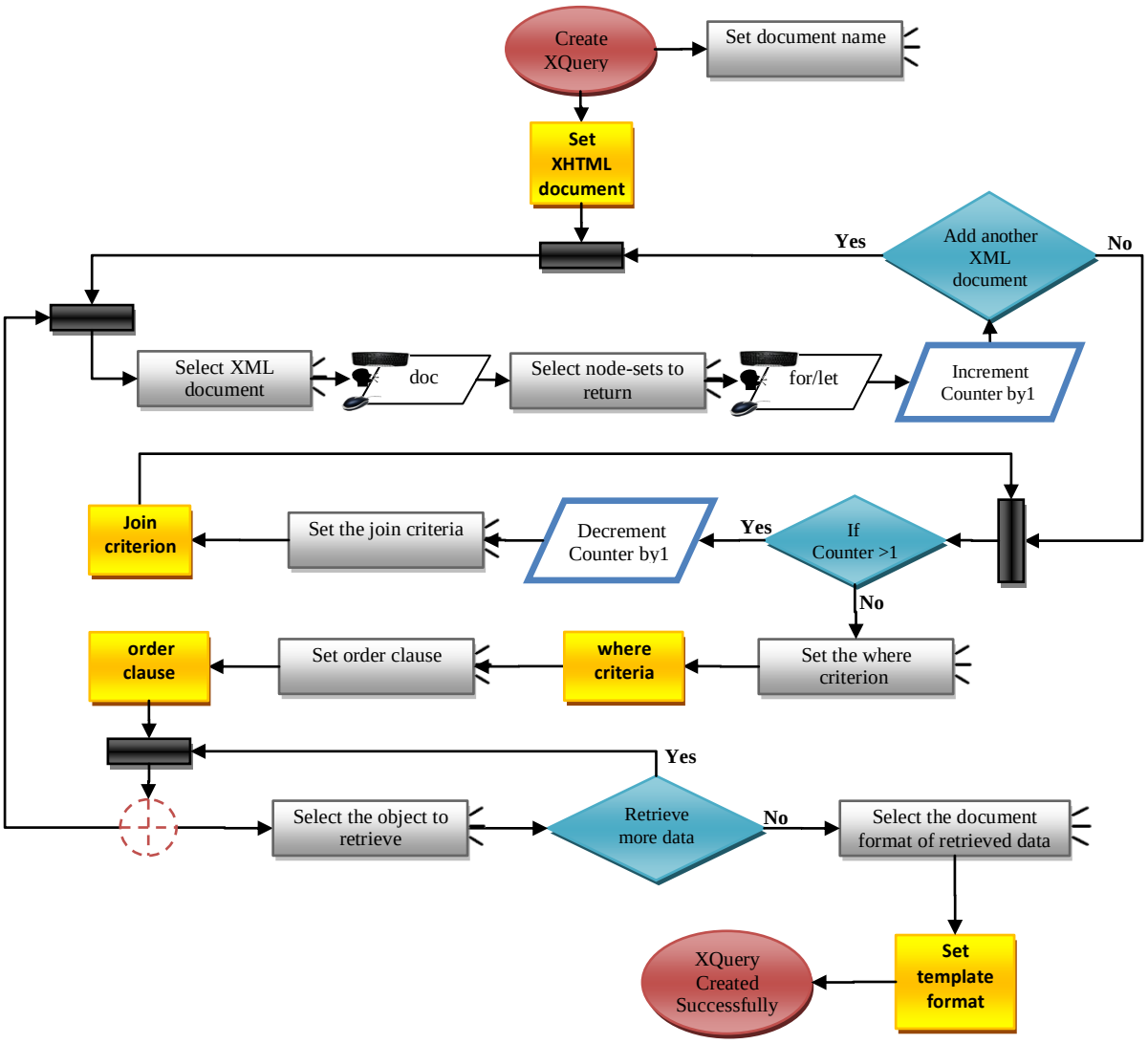

Figure (3.13): XQuery Creation Flow Chart

As shown in the previous diagram, creating XQuery process starts by setting the document name to be created. The document name includes the path and the extension of the file. The developer has the option to replace or create a new file; the file type can be either XML or XHTML of extension ".xml" or ".xhtml" respectively. After setting the file name, the user is prompted to specify the XML source document, and the node-set to be defined using the "for" or "let" clause, "for" tag can be used to define a variable to iterate within the list of nodes, while let return a list of all nodes to the variable, multiple node-sets can be set-in separated by vertical bar "|", where all specified nodes are retrieved. To enable the user to join data from several XML documents, one or more xml document can be selected within the same XQuery, then the user is prompted to set the join criteria between each two XML documents; in order to join the shared key for each two XML documents. Next step is to set the "where" clause; the criteria used to retrieve the data , one or more

condition can be set-in using the where clause, separated by logical operator, the where condition is a Boolean expression of two comparative operands and operator such as “and”, “or” , “<”,“>”,”<=”,”<=” , “!=“, …etc, each operands can be either element name, attribute name, constant value, calculated value or even XQuery function. Then the user is prompted to set the “order” clause, which also can be element name, attribute name, constant value, calculated value or even XQuery function, followed by the sorting criteria which can be either ascending or descending. Multiple objects can be set-in in the order clause, separated by commas. The next step is to specify the result of the XQuery statements, by setting its “return” clause, one or more data element, attribute or function can be specified in this portion, separated by commas , or even another FLOWR expression can be returned, in order to create nested queries, nested quires can be useful to create hierarchal xml document. Finally, the user has to select the document format of the retrieved data which can be either in XML or XHTML document. The following diagram shows the criterion used to return the result of the XQuery in XML document:

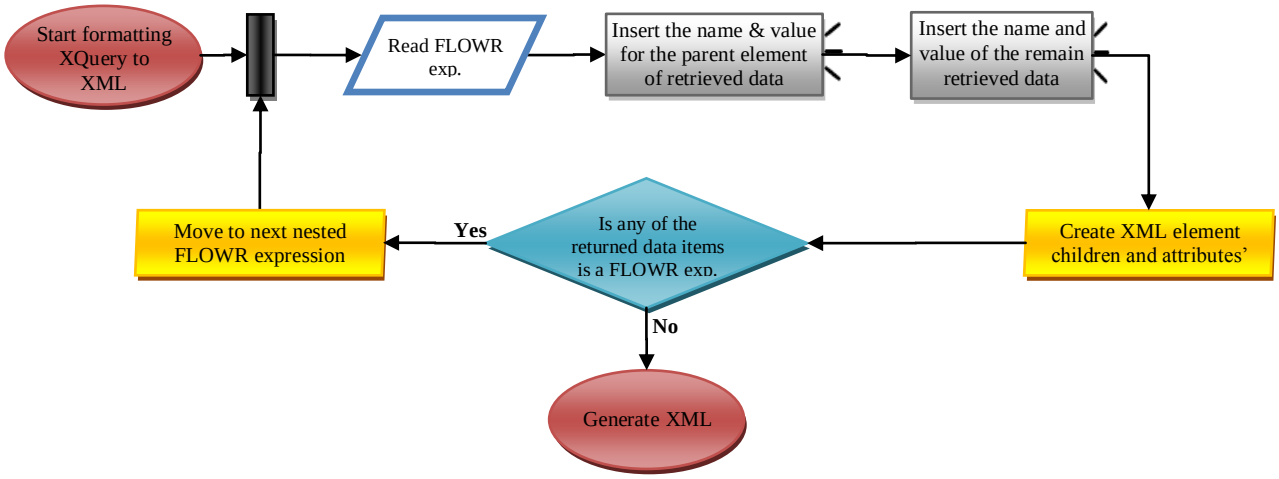

Figure (3.14): XQuery-to-XML Flow Chart

As shown in the above diagram, the XQuery results can be retrieved in either an XML or XHTML document, if the user wants to retrieve the results in XML Document, then for each FLOWR expression, the user has to insert the parent element name and value and attributes or elements list that will identify the returned data items, by default the elements and attributes names are set-in as defined in the original XML document, but user can modify it if needed. Only one parent element can be selected for each query level, if any of the returned data is another FLOWR expression, then it will be inserted as nested node

within the parent element, and will have its own attribute and elements the same way as its parent.

As illustrated in the following diagram , Retrieving XQuery results in XHTML format is also applicable, the <html>, <head>, <title>, <body> tags are added automatically, then the user is prompted to insert the title of the page, then the user is prompted to set the referred styling sheet in the "href" attribute of the link tag <link> or to set the format of each returned data item, whether to show data in bold <b>, italic <i>, underlined <u>or header<h1>...<h6> format, in addition to its font type and size and color. Then he has the option to retrieve the XQuery results as either a paragraph using <p> tag , or in a table using <table> tag or as ordered or unordered list using <ol> or <ul> tags , and he can insert a result title, in addition he can insert a header or label for each returned data item. For each query in the XQuery document, the user can present the results in different format according to his request, for example a table can have nested list or paragraph with different color or font. Our report now is ready and can be referenced from any application through button, hyperlink or when submitting a form.

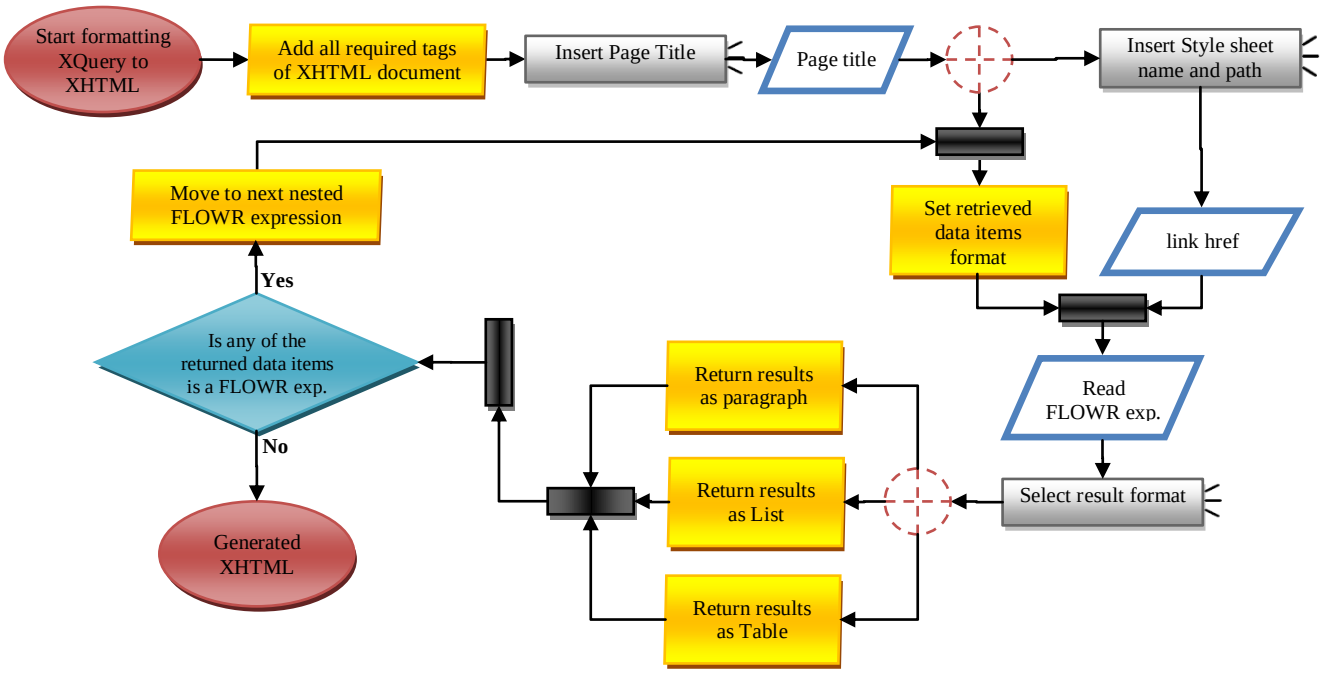

Figure (3.15): XQuery-to-XHTML Flow Chart

Chapter 4

Development & Implementation:

In our development and implementation of the pilot project of XMLBB tool, we build it using the visual basic dot net programming language, under the visual studio 2008 suite, framework 3.5. We create *blind_accessibility* package that can be imported from any windows form to automatically apply the accessibility methods previously discussed in section 3.2 such as enlarging form and speech-driven commands. Also we create a *template_form* that can be inherited in any form to apply others accessibility methods such as keyboard shortcut to access specific form control.

To retain the system and user data, we use XML documents. And to enable multi-lingual capability in our application, none of the interaction data, messages, error or control properties that may displayed to the developer are hard-coded; all are retained in XML documents.

This pilot project is a desktop application that can be used by blind programmers to create XML schema document, which identifies the structure of the XML database, and the data type of database elements and attributes, and their restriction and hierarchy. It's built over dialog forms, and speech-driven commands.

when all XMLBB component are implemented, The proposed output of this tool is XHTML web page, that includes XForms to input user data, JavaScript to validate user input and play speech messages to the end user in a specific events, SALT to define the prompt elements of speech text to the end-user, XPath to traverse through the XML document, form actions that are executed in specific XML events. In addition to summary

reports in XML or HTML format to provide user with specific information according to his request.

4.1 System Requirement:

XMLBB requirement differs according to the production stages. Whether we are in implementation or deployment stage; also the deployment stage itself can be either the XMLBB tool deployment or the end-user XMLBB-generated application deployment. The following subsections illustrate the system requirement for implementing and deploying all XMLBB components.

4.1.1 Development Requirements:

Several software are used in the development of the XMLBB tool; the application, and accessibility package are created using Visual Studio 2008 suite, under either windows 7 or Vista operating system. IIS web server and windows speech recognition feature need to be installed and enabled. And "System.Speech.dll" (version 3) library is used to create speech-enabled desktop application.

While for the testing the XMLBB-generated web application, speech software development kit SpeechSDK 1.1 is required. This package is used to test the speech technology in the generated web pages which is augmented with Speech Application language tag (SALT). Since XForms processor are not yet supported in web browsers, FormsPlayer plug-in can be used to process the XForm. The XMLBB is developed using VB.NET, XML documents, speech library, SALT, XHTML and JavaScript.

4.1.2 Deployment Requirements:

The deployment of XMLBB tool requires several services and software's to be installed in both client side and server side. In client side, windows speech recognition feature have to be enabled, XForms processor plug-in have to be installed, in addition to Microsoft .Net framework 3.5.

In the server side IIS web server, and speech server 2004 are required. We suggest speech server 2004 since latest version of speech server does not support the development of multi-model application. Only the development of telephony applications is possible.

While the deployment of XMLBB-generated web application requires the hosting of the application in IIS web server and Speech server 2004 in server side, in addition to XForms processor in client-side.

4.2 Database Structure:

In XMLBB development, we use XML documents to retain the system and user data. For the accessibility methods package, we created a data access layer that consists of two XML documents; "Messages.xml" to retain the *id*, *type* and *description* of each system message that may be displayed to the user, the *id* is unique identifier for the message, the *description* is the message itself and the *type* used to specify whether the message is a notify, error, caution or command message.

"Shortcuts.xml" document which used to keep the list of keyboard shortcuts that the user can use to run specific action, it consists of several elements; *Id* is a unique identifier, *char_code* retains the shortcut code, *shortcut_desc* used to set the description of the action of the shortcut, *keys* identify the keys that can be pressed to execute the action and *enabled* used to determine whether the action is in enabled or disabled state. Following are sample of the data that is retained in this XML documents:

Messages.XML

```
<?xml version="1.0" encoding="utf-8"?>
<shortcuts>
  <shortcut>
    <id> 1</id>
    <char_code>
      16
    </char_code>
    <shortcut_desc>
      pronouncing keys
    </shortcut_desc>
    <keys>
      Ctrl + "P"
    </keys>
    <enabled>0</enabled>
  </shortcut>
 </shortcuts>
```

Figure (4.1): Messages.xml document snapshot

Shortcuts.XML

```
<?xml version="1.0" encoding="utf-8" ?>
<messages>
  <message>
    <id>
      4
    </id>
    <description>
      can't enlarge more , reached the maximum size allowed
    </description>
    <type>
      1
    </type>
  </message>
</messages>
```

Figure (4.2): Shortcut.xml document snapshot

For the XMLBB interface, we created three XML document; the "Forms_Access.xml", "Forms_Notification" and "User-Projects". Following snapshot shows example of the details return for "XMLBB-Interface" form

Forms_Access.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<forms>
<form name="XMLBB_Interface" AccessibleDescription="XMLBB interface is the main
page of XMLBB; it's can be used by a user to create his/ her own web project easily
and efficiently , user can either create , or modify or delete a project using
this/ her page" AccessibleName="&amp;Interface" Text="XML Builder for Blind
Programers Interface">
     <controls>
    <control name="New_Project">
      <AccessibleDescription >
        creating a new web project
      </AccessibleDescription>
      <AccessibleName>
        &amp;Create
      </AccessibleName>
      <Text>
        Create New Project
      </Text>
    </control>
   <control name="Modify_Project">
    <AccessibleDescription>
      Modify Existing Web Project
    </AccessibleDescription>
    <AccessibleName>
      &amp;Modify
    </AccessibleName>
    <Text>
      Modify Existing Project
    </Text>
  </control>
</controls>
</form>
</forms>
```

Figure (4.3): Forms_Access document snapshot

The Forms_Access.xml document holds a detailed accessible description of each form and all its controls, and is used by the system to provide user with help concurrent declarative and help message. This XML document has been used in our system to enables the option of supporting multiple languages, i.e. enable the tool interaction to be in both English and Arabic. It consists of several elements and attributes; the *form* element which holds the form name in *name* attribute, *AccessibleDescription* attribute that is used to show the objective of the form, the *AccessibleName* attribute to identify the accessible name of the form, in addition to *text* attribute that retain the form title. It also contains a list of sub-elements; that retains the *Name* attribute and *AccessibleDescription*, *AccessibleName* and *Text* sub-elements of all form controls.

The Forms_Notifcation.xml document holds the list of *errors* and notification *messages* in the application. For each *Notification* sub-element, the *id* and the *type* attributes of both are defined, the type specifies the error or message type to be displayed, i.e. whether the error shows a specific field in the form is required or shows a custom error, or determines the range of the values that can be inserted in this field for the error element, or it confirms record deletion or notifying the completion of a process successfully. The following are sample of data retained in this document:

Forms_Notifcation.xml

```
<?xml version="1.0" encoding="utf-8" ?>
	<Notification>
		<Errors>
			<Error Id="1" Type="Required">
				The F-1 field is required , Can't be empty
			</Error>
		</Errors>

		<Messages>
			<Message Id ="13"
			Type="NotifySchemaElementCreateSuccessfuly">
			New schema simple type have been created
			successfully
			</Message>
		</Messages>
	</Notification>
</xml>
```

Figure (4.4): Forms_Notification.xml document snapshot

Finally the User_Projects.xml document, which retains the list of user-created project's .This document, consists of project schemas within the *schema* sub-element, and a list of project forms and reports inside *forms* and *reports* sub-element. Since our pilot project objective is the process of creating application schemas, here we specify only the details of the *schema* sub-element.

The *project* element contains the project *name*, which retain the project name and path that have to be unique in the system level, also it has *details* sub-element that have several attributes; the project default *namespace*, in addition to the *schema*, *forms* and *reports* folders names.

While the schema element has the *name* attribute, in addition to one or more sub-elements named with the user-created schema sub-elements. Each schema sub-element specifies the *object_type*; that is whether the schema sub-element is of type element, attribute, simple or complex, the data-type which can be built-in, simple element name or complex element name.

It also keeps the *default* and *fixed* value of the schema object, whether it's *nillable* or not. What to do with *whitespace. Use* attribute for attribute object to define whether this attribute is optional or complex. Also define the restriction criteria that can be applied to simple elements, such as e*numeration* list, text *pattern, minvalue , maxvalue, minlength , maxlength, length, fractiondigits and totaldigits* . In addition to *order* attribute in complex elements to indicate how sub-elements of the complex element can be ordered, and a list of sub-elements that include the *ref* attribute which specifies the name of the sub-elements that constitute the complex element.

The following are example data retained for a user project created using XMLBB:

User_Projects.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Projects>
  <Project Name="C:\NIIT_Project">
    <Details Namespace="www.ni-it.org" Schema="Schemas" Forms="Forms"
Reports="Reports"></Details>
    <Schemas>
      <Schema Name="NIIT-Database.xsd">
         <City object_type="element" data_type="string" whitespace="Preserve"
enumeration="Ramallah,Jerusalem,Jericho, hebron, other"  default="Ramallah"/>
         <Street object_type="element" data_type="string"
whitespace="Preserve"/>

        <Email object_type="simple" data_type="string"
       pattern="[^@]+@[^@]+\.[^@]+"/>

        <Mobile object_type="attribute" data_type="Integer" use="Optional" />

        <Address object_type="complex" order="sequance" >
             <element ref=”City”/>
             <element ref=”Street”/>
             <element ref=”Email”/>
             <element ref=”Mobile” minOccurs="1"//>
       </Address>

        <Name object_type="element" data_type="string" nillable="False" />

        <Rate Object_Type="Element" Data_Type="decimal" Nillable="False" />

        <Trainer object_type="complex" data_type="string" order="sequance" >
             <element ref=”Address”/>
             <element ref=”Name”/>
             <element ref=”Rate”/>
       </Trainer>

      </Schema>
    </Schemas>

    <Forms></Forms>

    <Reports></Reports>
  </Project>
  </Projects>
```

Figure (4.5): User_Projects.xml document snapshot

4.3 System Libraries:

In the development of XMLBB, accessibility package has been created, which is designed specifically to overcome the difficulty blind people faces in using GUI. It enables them to use their speech and keyboard shortcuts to create integrated web application. “Blind_Accessibility.vb” package can be imported in any dot net windows form to apply accessibility methods to this form. Blind_accessibility package consists of six classes;

Show_content, *execute_shortcuts*, *help*, *recognize*, *enabling_commands*, *enlarge*, and each class consist of several procedures and functions.

Blind_Accessibility package is the main class that instantiates all other classes and applies the accessibility facilities in the active form. It imports two namespaces, System.Speech.Synthesis package used to convert text to speech, and System.Speech.Recognition package which converts speech to text. The new constructor *new* set the form default button; which is executed by default when pressing "PageUp" key or by pronouncing execute. In addition, it maximizes the form to screen size, and enables form auto scrollbar so that if it is enlarged more than screen size, a scrollbar is shown automatically.

Show_content class main objective is to play the forms controls contents to speech using Microsoft *System.Speech.Synthesis* package; which helps the developer to recognize the content of the current control, and consequently ensures its correctness. It consists of several procedures and functions; the *Speak_control_text* procedure speaks the whole text value of the active control if it is not empty; regardless of the control type. While *Speak_first_word() and Speak_last_word()* procedure speaks the first and last word of the text respectively. And *spell_text* procedure to spell the active control text character by character. *say_char* procedure is used to pronounce each character the developer pressed, and *Show_msg_async*, and *Show_msg* procedures both can be used to display help, error and other type of messages to the developer in both text and speech, but the first one plays message asynchronously; so during playing the message another event can be done at the same time.

execute_shortcuts class is used to execute specific action when user presses keyboard shortcut or pronounce a specific command. In its constructor, it adds two events to the form, *keypress* and *keydown*, and two procedures that implement these events; *form_KeyPress* first check whether the pressed key is a shortcut or not, this done by checking its existence in *shorcuts.xml* document using *check_if_shortcut(keycode)* function, if it exist, according to the pressed shortcut, the form runs a specific action, for example it may enables or disables pronouncing pressed key, displays a help message to the user, enlarges or reduces the form size, notifies the user with the active control or even executes any method in the show-content class.

While *form_keydown*, check whether the user presses either *PageUp*, *F1*, *F2* or *Delete* key; if *PageUp* key is pressed then it will execute the form default action, by using *Execute_default_action* method which simply call the *performclick* method of the default button, while *delete* key check if the content of the active control could be deleted or not; depending on its delete tag property, if yes it clears this control content. Finally, pressing F1 or F2, provides the user with form or project description respectively using help class.

Help class aims to provide user with help when requested. It has three methods, *Speak_help_message*, *from_help* and *project_help*. The *Speak_help_message* procedure provides the user active control accessibility description, in addition to the current value of the control, and the selected value if list controls is used. *from_help* procedure displays help in form level , and project_help method display a help message about the active project.

Recognize class is very similar to *execute-shortcut* class in its objective. It listens to the user speech in order to extract a specific command. It consists of several procedures and functions; *createGrammer* is a primary procedure which specifies the grammar of the user speech. And *spRecognizer_SpeechRecognized* procedure which recognizes user speech defined in this grammar, and then executes the action identified for this command. For example, when user says "Read Text", the active control text will be played into a speech.

enabling_commands is a very simple class , it just either enables or disables or changes the shortcut command state. It consists of three procedures; *Enable_Cmd* , turn the shorcut enabled state in shorcuts.xml document to true regardless of the current state, and notifies the user that the shortcut is enabled, *Disable_Cmd* on the other side disables the shortcut, and notifies the user. While *Change_status_cmd* , changes the enabled state from true to false and from false to true, and notifies the user with the new state.

The last class is *enlarge* class. *enlarge* class enable visually impaired people to enlarge and reduce the form size according to their request. It changes the size of every control on the form, and its content, and automatically added a scrollbar to it, so that user can traverse all controls in the form. It consists of four procedures*; enlarge*, *Max_enlarge*, *Reduce* and *Min_Reduce*. *Enlarge* doubles the form size, and Max_enlarge doubles the form size four

times of the original size. *Reduce* minimizes the form size to half, and *Min_Reduce* reduces the form size to one over eight of the original size.

4.4 Application Interface:

In our development of XMLBB interface, all forms are enhanced with several accessibility methods by instantiating the *blind_accessibility* package, and providing its constructor with the form and its default button parameters, as well as inheriting the *template_form*. *template_form* fills the accessibility and text properties of the form and its controls from the *forms_access.xml* document at form load event. It also displays the form title and controls names, by both speech and text on received focus event, and manipulates generic control validation, if defined in form tag property, such as *required* validation. It automatically displays an error message to the user when attempt to leave the control empty. In case of custom validation; it is manipulated locally at the form level. And finally it registers the hot keys for each control in the form.

In XMLBB interface, if the windows speech recognition is opened, the user can enable or disable speech listener by saying "start listening" or "stop listening" respectively. Then he may use his speech to execute actions in the form.

In every form in the XMLBB, user can recognize what are the type, value and objective of each control in the form when received focus. Also he can request for more help in several levels; in the form level by pressing F1 or saying "form help", in the control level by pressing "Ctl+C" or saying "control help", or in the project level by pressing F2 or saying "project help". He may also move up and down through the controls by pressing "Tab" or "Shift+Tab", or by saying "next" or "previous". He may also press "Alt+E" or say "exit" to exit the form.

User can jump directly to a specific control in the form by either pressing "Alt" plus specific character or by saying the control name, noting that accessing a control directly using the hot keys executes the default action of the control.

All forms in XMLBB can be accessed by using the previous criteria .Following are few examples of the XMLBB forms, their detailed description, and their objectives:

XMLBB_Interface form shown below, is the main page in XMLBB tool, through this form, user can create new projects, modify and /or delete existing projects.

When user initially runs the XMLBB tool, this form is opened, and the title of the form is displayed in both text and speech. This can help the user to recognize the objective of this form. It also displays the name and type of the active control in the form; in both speech and text, which is "create" button in this form. If the user requests more details about this form, he may either press F1, or by saying "form help".

User also can request help in control level by either saying "help" or pressing "Ctl+C", and it will display the description of the control, and its content when available, prefixed with control type, in our case "button". For novice user, he can recognize the form controls by moving through all controls in the form and requesting help in each level. He can either say "next" or "previous" or press "Alt" or "Shift+Alt" to traverse the form control's.

For an expert user, he can jump and execute the default action of a specific control directly. For example, to create a new project, user can either say create, or press "Alt+C" keys, and *on-click* event will be executed. User may exit the form by saying *exit,* or by pressing "Alt+E", or by moving to the Exit button, by mouse, keyboard or speech and then press enter. Also he can execute the form default action by saying "execute form", or pressing "PageUp".

If user is visually impaired, he can enlarge the form size so that he can read and access it more easily. This can be done by either saying "enlarge" or "enlarge maximum", or by pressing "Ctl+E" or "Shift+'+'", or reducing form size by saying "reduce" or "reduce minimum", or pressing "Ctl+R" , or "Shift+'-'".

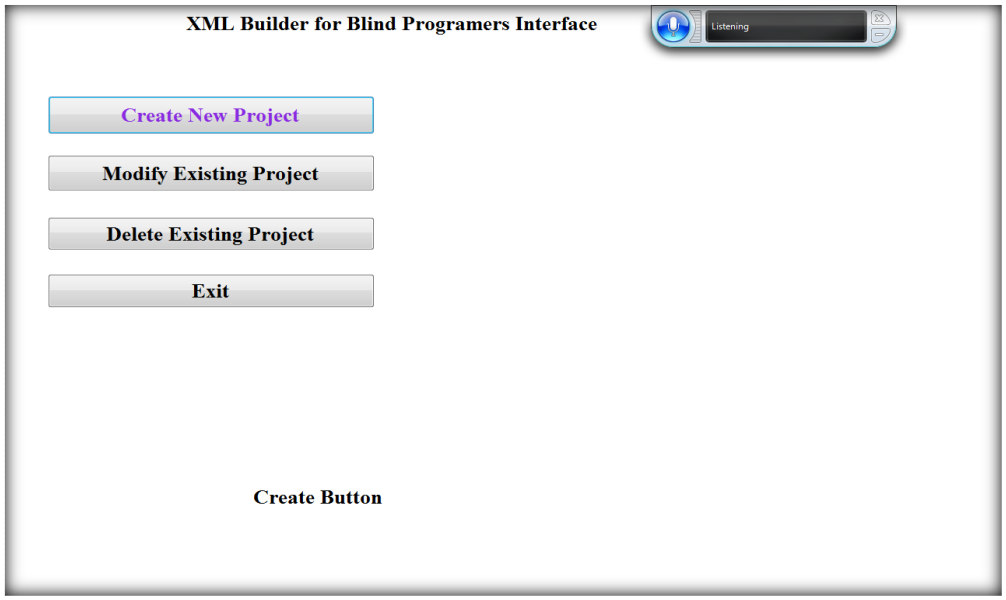

Figure (4.6): XMLBB Interface

If user wants to create a new project, he can either says "create", or press "Ctl+C" or moves to the create button and click it and/or presses enter key. Then the *New_Project* form is opened. Following is a snapshot of this form:

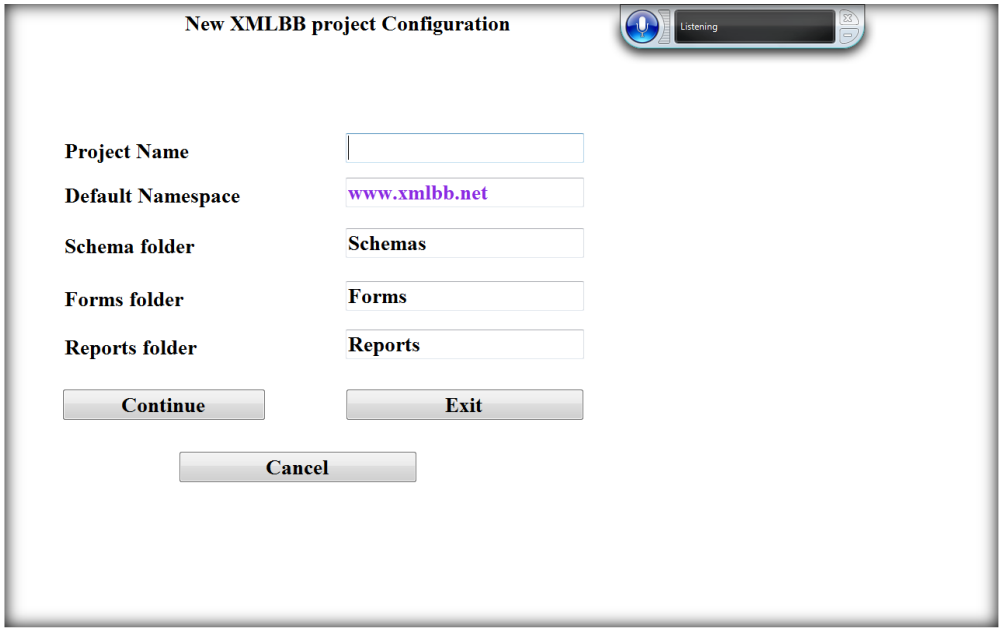

Figure (4.7): New Project Configuration

The *New_project* form can be used to set the configuration property of the new project; the name and path of the project, its default namespace and the name of the folders for schemas, forms and reports. When the *New_project* form is opened, the form title is displayed, and the cursor moves to the first control on the form, i.e. the "project" textbox,

and then the name and description of the control is displayed in both speech and text to help the user to recognize the type and format of the data he have to input.

For novice users, who are not familiar enough with keyboard keys, they have the option to enable or disable pronouncing pressed characters by saying “enable pronouncing”, or “disable pronouncing”, or by pressing “Ctl+P”.

In *New_project* form the user can access any control directly by saying the control name, or by pressing “Alt+P”, “Alt+N” “Alt+S”, “Alt+F”, “Alt+R” to jump to project, namespace, schemas, forms or reports textbox respectively. In this form, since all these fields are required, leaving any empty field is not allowed and an error message will be displayed to the user to notify him that the field is required.

Committing changes in each form is done automatically when moving to the next one in the wizard. When closing any XMLBB form, a request for confirmation message will be displayed to the user to confirm exiting the form, followed by another message to confirm his interest of committing changes. In this form, two new navigation controls have been added, “Okay” and “Cancel” buttons, where “Okay” moves the user to next step of creating new form wizard, while cancel button exits the wizard. If the user says “Okay” or press “Alt+O”, then a message is displayed to user to notify Him that his form is created successfully, and the *Project_Objects* form is opened.

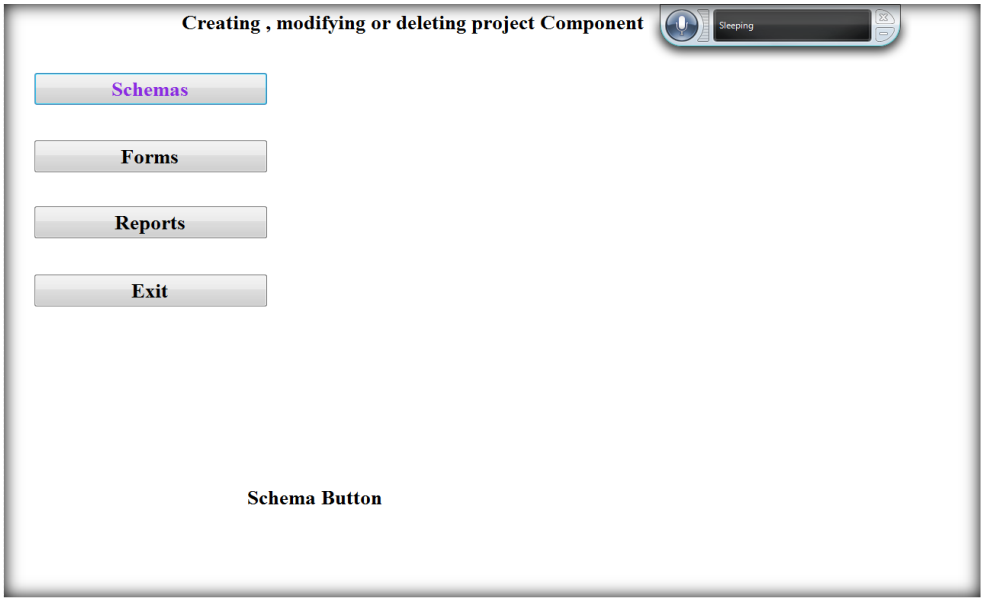

Figure (4.8): Creating/Deleting or Modifying Project Component

Project_Objects form enables the user to create, modify or delete project component; i.e. user may create multiple project schemas, forms and reports. Since the user can access this form through creating or modifying existing project, pressing “Schemas”, “Forms” or “Reports” buttons can forward the user to different forms according to the current state of the project. If project is new then pressing schema buttons will forward the user to *new_schema* form, but in case, at least one schema document is created to the current project, then the user is forwarded to *Project_Schemas* form. In our pilot project, only creating project schema is implemented, so the “forms” and “reports” buttons are disabled.

Since XMLBB is based on wizards, it consists of many forms; each is used to fill specific and simple information. But all follow the same accessing and using criteria. When the user gets used to any form he will be able to use all others easily.

Chapter 5

XMLBB Training, Testing & Evaluation:

In the development and implementation of XMLBB tool, it was not easy to anticipate entirely what the blind people would consider more suitable for their needs, and what they expect from the XMLBB tool. So we conducted four sessions, about three hours each to train our blind volunteer on the tool. The main objective of these sessions is to test the tool performance and usability and to refine, validate and evaluate it. Since we were unable to find a volunteers who has good knowledge of database concepts, the training sessions include a part of clarifying and explaining the database concepts using real test cases, how to apply these concepts using XML schema, and the usefulness of defining a valid database schema to the entire project. At the end of the training, the volunteer was able to build XML schema easily, and she also was able to manipulate with XMLBB tool and to totally depend on herself.

5.1 Training:

Because our blind volunteer has no database background, we started the training sessions by explaining the concept of database in general, and how to apply these concepts using XML schema. Several examples have been discussed, such as School and Supermarket database. Then we explained to her what schema objects can be used for building a database such as elements, attributes, simple types and complex types. Then clarified the differences between these objects, and what is best to use for each field in the XML database.

After this training, the volunteer was able to identify the object of the simple schema database, determine the schema objects and data type that fit her needs. Also she learned how to set the constraint on the XML database using the simple type objects, and how to group multiple schema objects using the complex type objects.

Next, we explained to her the objective of building XML schema, its effects on the form, size and type of data used in creating the XML document, and how the user can use it later to generate XForms which is used to collect data.

After explaining the database concepts, we trained her on how to use the XMLBB tool, and provided her with a list of shortcuts, hotkeys and speech commands that she can use to access XMLBB interface. Following are the entire list:

No.	Action	Shortcut	Speech Command
1	Enable or Disable pronouncing of every printable charctered user pressed	Ctrl + "P"	"Enable Pronounicng" or "Disable Prounouncing"
2	Display Help Message for the Active control	Ctrl+ "C"	"Control Help"
3	Enlarge Form Size	Ctrl + "E"	"Enlarge"
4	Reduce Form Size	Ctrl + "R"	"Reduce"
5	Enlarge Form Size	Ctrl + "U"	"Enlarge Maximum"
6	Reduce Form Size	Ctrl + "D"	"Reduce Minimum"
7	Display the Entire Text of the Active Control	Ctrl + "T"	"Read Text"
8	Display the Last Word of the Active Control Text	Ctrl + "L"	"Last Word"
9	Display the First Word of the Active Control Text	Ctrl + "F"	"First Word"
10	Spell the Active Control Text	Ctrl + "S"	"Spell Text"
11	Display Form Help	"F1"	"Form Help"
12	Display Current Project Details	"F2"	"Project Help"
13	Jumb to the Next Control on the Ative Form	Up Arrow	" Previous "
14	Jumb to the Previous Control on the Active Form	Down Arrow	"Next"
15	Apply the Click Event of the Okay Button of the Active Form	Alt +"O"	"Okay"
16	Cancel the current process	Alt +"A"	"Abort"
17	Exit the Active Form and close XMLBB	Alt +"E"	"Exit"
18	Go back to previous page	Alt +"B"	"Back"
19	Jumb Directly to a Specific Control	Alt + 1st or 2nd character (control Text)	Pronounce the first word of the control text
20	Execute the default action of the form	PageUp	"Execute"
21	Stop the current playing speech	PageDown	"Stop"

Table (5.1): List of Shortcuts, Speech Commands & Hotkeys

To help the blind volunteer to become familiar with the XMLBB tool interface, we clarified to her in details how to use this tool by showing her all the accessibility methods in table 5.1.

5.2 Testing:

In the testing process, we suggest a usability testing methodology to evaluate the tool and add more improvement in it when necessary. But since we was unable to find qualified users for applying this part, we implemented this part without following our proposed testing methodology.

One blind volunteer expressed her wailings to examine the tool, but since she is specialized in a field that is quite different from computer technology, she was unable to evaluate the output of the tool, and whether the generated schema is valid in both syntax and semantic or not.

5.2.1 Proposed Testing Methodology:

Our proposed testing methodology consists of several stages during and after the tool development. First the XMLBB have to be tested by at least two expert users that have an experience in the XML schema in order to evaluate its functionality, and examine whether XMLBB cover all of the XML schema objects or not, and to identify possible problems in the tool. The feedback of the expert users determines whether to move to the next stage directly or to add more improvement in the tool.

The next step is to test the XMLBB tool by three to five blind end-users, who have a preliminary knowledge of database concepts. The end-users will follow three sessions, two hours each, to review their database knowledge, and how to apply all XML Schema concepts using this tool. In this stage the end-users interaction with XMLBB should be monitored in order to evaluate how simple they can access and interact with the tool and consequently improve it if necessary.

As a final step for the tool testing, the blind end-users are requested to design a specific database using XMLBB. And then expert user evaluate their created databases, and the end-users also write-down their opinion about usability and efficiency of the tool.

XMLBB Testing:

After four sessions of training, the blind volunteer becomes familiar with XMLBB tool. Since she is familiar with the screen reader software's such as Hal, but never used windows speech recognition technology before, and so she preferred to use keyboard shortcuts and hotkeys instead of speech commands.

In the testing process, our volunteer suggested divers comment and improvements to the XMLBB, based on her previous experience with the screen reader tools. She suggested displaying a speech message to the user to notify if any character is deleted. She also requested a notification message of the current character when moving within the textbox by arrows. She preferred to have the ability to stop the list of the playing speech messages by pressing a specific key. All her suggestions have been taken into consideration and we improved the tool according to it.

While monitoring her during the testing of the tool, we found that she faced some difficulties such as; when moving between controls using "Tab" key our volunteer was faster than the speech message that displays the active control name, so she thought by mistake that she is in the correct control. To handle this problem, we automatically stopped the previous played message when the cursor accessed the next control, so that both cursor and speech point to the same control.

Another difficulty she faced was that she is unable to recognize whether a new form is loaded or she is still in the same form. To handle this issue, we added a beep sound in the form load event that notifies the user when a new form is opened.

In the last session, the blind volunteer test the speech recognition commands, since she was unfamiliar with speech recognition technology from windows, she was not very satisfied of this technology, she said that it is easier for blind person to use either his speech or keyboard shortcuts to execute commands but not both.
Also she faced some difficulties because speech recognition is not very efficient since some speeches are interpreted incorrectly in textbox controls to "Home", which moves the cursor to the beginning of the control text while inputting data and causes invalid input to be inserted. To handle this problem, she stopped the speech listener whenever she wanted

to input any text. But at the same time, she said that using speech commands is interesting, and can be useful for disabled people who can't use their hands.

5.3 Evaluation:

When asking our volunteer about her opinion of the XMLBB tool; she was satisfied about it. She thinks that blind people can get used to it easily, since the help and declaration messages can be displayed to the user in any application level. Also she thinks that providing the user with several accessibility methods distinguish the tool, since every user can to use the accessibility method that he is comfortable with it.

Because she was unfamiliar with windows vista or windows 7, and since "Hal" screen reader does not support these operating systems, she suggested that XMLBB should be able to run in any operating system.

Also, our volunteer was pleased about the improvements we added finally to the tool, and she think that more testing with a larger number of blind users can help to enhance the tool more.

Also she thinks that this tool has to support other languages such as Arabic, since this will encourage more people to use this tool.

Chapter 6

Conclusion and Future Work:

When providing appropriate environment, blind people can compete with sighted people since they has distinctive capabilities that sighted people don't have, specifically in their memorizing capabilities. Our testing of the pilot project proves that a tool augmented with several means of enhancement can be used easily by blind people.

Developing blind oriented tools contributes in opening new job opportunities to blind people and consequently can improve their living conditions. The problem is not in the incapability of the blind people, but in lack of conviction of blind people capabilities from employers.

The integrated idea of our model becomes more obvious when all components of the XMLBB tool are created, specifically the XForm generation part, also this part can be enhanced with several templates, so that the user can either use the system built-in templates or custom one. Also many options have to be available to the user to customize his automatically generated interface, such as the height and width of the form controls, font type, size and color, and many others.

As we discussed in chapter 3, XMLBB have to support many languages to enable developers to create integrated web application, such as server-side and client side scripting languages, in addition to styling sheets and others.

Since the XMLBB is implemented to support multilingual, one of the main objectives in the future is to use speech recognition and synthesizers technology

that supports Arabic languages, to get rid of the language barrier when using XMLBB.

The world of XML technology is constantly evolving, and new languages based on XML technology continuously emerges. Our model can be enhanced to support more XML-based programming languages, such as resource description framework (RDF) and the web ontology language (OWL).

One of the areas that become widely used in these days is the web services; also named XML web services, since it is based mainly on XML technology. Enhancing XMLBB with such feature can help blind people to expand their projects, and develop more professional and usable applications.

One of the weaknesses of speech recognition technology that should be enhanced is the dictating feature, either word by word, or character by character. Improving this can lead to a significant facilities in accessibility method for disabled people.

The development of accessible user interface is a never-ending process. There are always more features that can be added, which improves and facilitate the software usage.

For professional blind programmers, wizard is not very efficient; they may prefer to write code directly into editor. Providing a blind programmer with source editor should be very useful if augmented and enhanced with clear error and warning messages that identify the line and cause of the error.

References:

[1] Argyropoulos, S, Moustakas, K, Karpov, A, Aran, A, Tzovaras, T, Tsakiris, T, Varni, G, Kwon, B**.** (**2008**): Multimodal user interface for the communication of the disabled. J Multimodal User Interfaces 2: 105–116.

[2] B´artek, L, Plh´ak, J. (**2006**): WebGen System - Visually Impaired Users Create Web Pages. ICCHP 2008, LNCS 5105, pp. 466–473.

[3] Bigham, J, Aller, M, Brudvik, J, Leung, J, Yazzolino, Ladner, R. (**2008**): Inspiring Blind High School Students to Pursue Computer Science with Instant Messaging Chatbots **,** ACM 978-1-59593-947-0/08/0003.

[4] Franqueiro, K, Siegfried, R. (**2006**)**:** Designing a Scripting Language to Help the Blind Program Visually. ACM 1-59593-290-9/06/0010.

[5] Haraty, R, Ariss, O. (**2007**): CASRA+: A Colloquial Arabic Speech Recognition Application. American Journal of Applied Sciences 4 (1): 23-32.

[6] Kuo, Y, Shih, N, Tseng, L, Hu, H. **(2005)**: Generating Form-Based User Interfaces for XML Vocabularies. ACM 1-59593-240-2/05/0011.

[7] Najjar, L. (**2005**)*:* Accessible Java Application User Interface Design Guidelines. HCI International Proceedings.

[8] Sánchez, J, Aguayo, F. (**2005**): Blind Learners Programming Through Audio, ACM 1-59593-002-7/05-0004.

[9] Siegfried, R. **(2004)**: Teaching the Blind to Program visually. Proc ISECON v21.

[10] Siegfried, R. (**2006**): Visual Programming and the Blind: The Challenge and the Opportunity. ACM,1-59593-259-3/06/0003.

[11] Wald, M, Bain, K. (**2008**): Universal access to communication and learning: the role of automatic speech recognition. Univ Access Inf Soc , 6:435–447.

[12] W3C. (**2004**): XML Schema Part 0: Primer Second Edition. W3C Recommendation: http://www.w3.org/TR/xmlschema-0.

[13] W3C. (**2004**): XML Schema Part 1: Structures Second Edition. W3C Recommendation: http://www.w3.org/TR/2004/REC-xmlschema-1.

[14] W3C. (**2004**): XML Schema Part 2: Datatypes Second Edition.W3C Recommendation: http://www.w3.org/TR/xmlschema-2.

[15] W3C. (**2007**): XML Syntax for XQuery 1.0 (XQueryX). W3C Recommendation: http://www.w3.org/TR/xqueryx.

[16] W3C. (**2009**)*:* XForms 1.1. W3C Recommendation, http://www.w3.org/TR/xforms.

Appendix A: Source Code

---------- **Blind Accessibility Package** ----------

```
Imports System.Speech
'' Synthesis class convert text to speech

Imports System.Speech.Synthesis
'' reccognition class convert speech to text

Imports System.Speech.Recognition
'' schemas in data access layer

Imports XMLBB.Access
'' accessibility

Public Class Blind_Accessibility
    'Inherits System.Windows.Forms.NativeWindow

    '' synthesier used to say words
    Public Shared Syn As SpeechSynthesizer = New SpeechSynthesizer

    '' recognizer used to recognize the user speech
    Private rec As SpeechRecognizer = New SpeechRecognizer

    '' used to define dictiation grammer to be able to turn it off or on
    Private DGrammar As DictationGrammar = New DictationGrammar()

    '' reference to messges class "Access Layer"
    Shared msgs As messages = New messages

    '' form
    Private frm As Form = New Form

    '' default button for default action
    Shared btn As Button

    '' create instance of help class
    Private hlp As help

    '' create instance of help class
    'Shared spk As Show_content = New Show_content

    '' create instance of enlarge class
    Shared en As enlarge

    '' create instance of show_content class
    Private cnt As Show_content

    Private Speech_Rec As recognize

    Public Sub New(ByVal myform As Form, ByVal default_Button As Button)

        frm = myform
        btn = default_Button
        hlp = New help(frm)
        en = New enlarge(frm)
        cnt = New Show_content(frm)
        Speech_Rec = New recognize(frm)

        ' set form window size maximized
```

```
        frm.WindowState = FormWindowState.Maximized

        'enable scrolling in windows form
        frm.AutoScroll = True


        '' create instance of execute shortcuts clas
        Dim exe_short As execute_shortcuts = New execute_shortcuts(frm)

    End Sub

        '' speech text in any control
    Public Class Show_content
        Private frm As Form
        '' synthesier used to say words

        Public Sub New(ByVal myform As Form)
            frm = myform
        End Sub

        ' thisfunction speak the text within a control to help user check
whether He is writting every thing correctly
        Public Sub Speak_control_text()

            If frm.ActiveControl.Text.Length = 0 Then
                Syn.SpeakAsync(msgs.get_message_byID(10))
            Else
                Syn.SpeakAsync(frm.ActiveControl.Text)
            End If
        End Sub

        ' thisfunction speak the text within a control to help user check
whether He is writting every thing correctly
        Public Sub Spell_control_text()
            Dim shrt As shortcuts = New shortcuts
            If frm.ActiveControl.Text.Length = 0 Then
                Syn.SpeakAsync(msgs.get_message_byID(10))
            Else
                For Each text_char In frm.ActiveControl.Text
                    If Char.IsLetterOrDigit(text_char) Then
                        If Char.IsUpper(text_char) Then
                            Syn.SpeakAsync("Capital " + text_char)
                        Else
                            Syn.SpeakAsync(text_char)
                        End If
                    Else
                        Syn.SpeakAsync(shrt.get_KeyName(text_char))
                    End If
                Next
            End If
        End Sub
        'pronounce the character "speech"
        Public Sub pronounce_char(ByVal text_char)
            Dim shrt As shortcuts = New shortcuts
            If frm.ActiveControl.Text.Length = 0 Then
                Syn.SpeakAsync(msgs.get_message_byID(10))
            Else
                If Char.IsLetterOrDigit(text_char) Then
                    If Char.IsUpper(text_char) Then
                        Syn.SpeakAsync("Capital " + text_char)
                    Else
```

```
                    Syn.SpeakAsync(text_char)
                End If
            Else
                Syn.SpeakAsync(shrt.get_KeyName(text_char))
            End If
        End If
    End Sub
    'return the character "speech"
    Public Function Return_char(ByVal text_char) As String

        Dim shrt As shortcuts = New shortcuts
        If frm.ActiveControl.Text.Length = 0 Then
            Return (msgs.get_message_byID(10))
        Else
            If Char.IsLetterOrDigit(text_char) Then
                If Char.IsUpper(text_char) Then
                    Return ("Capital " + text_char)
                Else
                    Return (text_char)
                End If
            Else
                Return (shrt.get_KeyName(text_char))
            End If
        End If
    End Function

    ' thismethod can be used to say the Last word of contol  text
    Public Sub Speak_last_word()
        '' active control
        Dim control_text As String = frm.ActiveControl.Text
        '' last word
        Dim Last_word As String =
control_text.Substring(IIf(control_text.LastIndexOf(" ") <> -1,
control_text.LastIndexOf(" "), 0))
        If Last_word.Length = 0 Then
            Syn.SpeakAsync(msgs.get_message_byID(10))
        Else
            Syn.SpeakAsync(Last_word)
        End If
    End Sub
    ' thismethod can be used to say the first word of contol  text
    Public Sub Speak_First_word()

        Dim control_text As String = frm.ActiveControl.Text
        ''first word
        Dim first_word As String = control_text.Substring(0,
IIf(control_text.IndexOf(" ") <> -1, control_text.IndexOf(" "),
control_text.Length))

        If first_word.Length = 0 Then
            Syn.Speak(msgs.get_message_byID(10))
        Else
            Syn.SpeakAsync(first_word)
        End If
    End Sub

    ' say the charcter pressed to help user ensure He is pressing the
correct character
    Public Sub say_char(ByVal chr As Char)
        Syn.Speak(chr)
    End Sub
```

```
        '' spoken & written
        Public Sub Show_msg_async(ByVal msg As String)
            If Not (frm Is Nothing) Then
                frm.Controls("msg").Text = msg
            End If
            Syn.SpeakAsync(msg)
        End Sub
        Public Sub Show_msg(ByVal msg As String)
            If (Not (frm Is Nothing)) And (Not (frm.Controls("msg") Is
Nothing)) Then
                frm.Controls("msg").Text = msg
            End If
            Syn.Speak(msg)
        End Sub
    End Class

    Public Class execute_shortcuts
        Private frm As Form
        Private cnt As Show_content
        Private hlp As help
        '' synthesier used to say words

        Public Sub New(ByVal myfrom As Form)
            'indicate that the form will receive key events before the
event is passed to the control that has focus
            frm = myfrom
            cnt = New Show_content(frm)
            hlp = New help(frm)
            frm.KeyPreview = True
            ' enable key press event to be able to handle it
            AddHandler frm.KeyPress, AddressOf form_KeyPress
            AddHandler frm.KeyDown, AddressOf form_keyDown
        End Sub

        '' work whenever a user press a key in keyboard
        Private Sub form_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs)
            Try
                Dim shrt As shortcuts = New shortcuts

                'enabling_commands class to enable or disable commands
                Dim cmds As enabling_commands = New
enabling_commands(frm)

                Dim code As Int16 = Asc(e.KeyChar) ' return the asci code
of the keys pressed


                Dim p_enabled = shrt.check_shortcut_enabled_byID(1) '
check if pronouncing is enabled

                If Not shrt.check_if_shortcut(code) Then ' check if code
of the keys pressed is not shortcut
                    If p_enabled And (frm.ActiveControl.GetType.ToString
= "System.Windows.Forms.TextBox") Then '' pronouncing key enabled
                        If Char.IsLetterOrDigit(e.KeyChar) Then
                            cnt.say_char(e.KeyChar)
                        Else

cnt.Show_msg_async(shrt.get_KeyName(e.KeyChar))
                        End If
```

```
                    End If
                Else
                    ' if key pressed is a shortcut
                    If code = 16 Then
                        ' pronouncing key shortcut
                        cmds.Change_status_cmd(1, 2, 3) ' 1- shortcut
id=1 "pronouncing key" , 2- message id =2 "pronouncing key is enabled" 3-
message id =3 "pronouncing key is disabled"
                    ElseIf code = 8 Then
                        ' help shortcut
                        If Me.frm.ActiveControl.GetType.ToString =
"System.Windows.Forms.TextBox" Then
                            Try
                                Dim txt_ctl As TextBox =
CType(Me.frm.ActiveControl, System.Windows.Forms.TextBox)
                                If txt_ctl.Text.Length > 0 Then

cnt.Show_msg_async(cnt.Return_char(txt_ctl.Text.Substring(txt_ctl.Selecti
onStart - 1, 1)) & " deleted")
                                Else
                                    cnt.Show_msg("Text Box is Empty")
                                End If

                            Catch
                            End Try
                        End If
                    ElseIf code = 5 Then ' Enlarge shortcut
                        ' enlarging form to double
                        Syn.SpeakAsync(msgs.get_message_byID(7))
                        en.enlarge()
                    ElseIf code = 18 Then ' reduce shortcut
                        ' reducing form size to half
                        Syn.SpeakAsync(msgs.get_message_byID(8))
                        en.Reduce()
                    ElseIf code = 19 Then
                        ' say active control text shortcut
                        cnt.Spell_control_text()
                    ElseIf code = 20 Then
                        ' say active control text shortcut
                        cnt.Speak_control_text()
                    ElseIf code = 12 Then
                        ' say text of active control( last word) shortcut
                        cnt.Speak_last_word()
                    ElseIf code = 6 Then
                        ' say text of active control (first word)
shortcut
                        cnt.Speak_First_word()
                    ElseIf code = 21 Then ' enlarge maximum
                        ' enlarging form to maximum size
                        Syn.SpeakAsync(msgs.get_message_byID(7))
                        en.Max_enlarge()
                    ElseIf code = 4 Then ' reduce minimum
                        ' reducing form size to minimum size
                        Syn.SpeakAsync(msgs.get_message_byID(8))
                        en.Min_Reduce()
                    End If
                End If
            Catch ex As Exception
                cnt.Show_msg_async(ex.Message)
            End Try
        End Sub
```

```
        Private Sub form_keyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs)
            Try
                If e.KeyCode = Keys.Right Then
                    If Me.frm.ActiveControl.GetType.ToString =
"System.Windows.Forms.TextBox" Then
                        Try
                            Dim txt_ctl As TextBox =
CType(Me.frm.ActiveControl, System.Windows.Forms.TextBox)
                            If txt_ctl.Text.Length = 0 Then
                                cnt.Show_msg_async("Last Character")
                            Else
                                If txt_ctl.SelectionStart <>
txt_ctl.Text.Length Then

cnt.pronounce_char(txt_ctl.Text.Substring(txt_ctl.SelectionStart, 1))
                                Else

cnt.pronounce_char(txt_ctl.Text.Substring(txt_ctl.SelectionStart - 1, 1))
                                    cnt.Show_msg_async("Last Character")
                                End If
                            End If
                        Catch
                        End Try
                    End If
                ElseIf e.KeyCode = Keys.Left Then
                    If Me.frm.ActiveControl.GetType.ToString =
"System.Windows.Forms.TextBox" Then
                        Try
                            Dim txt_ctl As TextBox =
CType(Me.frm.ActiveControl, System.Windows.Forms.TextBox)
                            If txt_ctl.Text.Length = 0 Then
                                cnt.Show_msg_async("First Character")
                            Else
                                If txt_ctl.SelectionStart <> 0 Then

cnt.pronounce_char(txt_ctl.Text.Substring(txt_ctl.SelectionStart - 1, 1))
                                Else

cnt.pronounce_char(txt_ctl.Text.Substring(txt_ctl.SelectionStart, 1))
                                    cnt.Show_msg_async("First Character")
                                End If
                            End If
                        Catch
                        End Try
                    End If
                ElseIf e.KeyCode = Keys.Delete Then
                    If Me.frm.ActiveControl.GetType.ToString =
"System.Windows.Forms.TextBox" Then
                        Try
                            Dim txt_ctl As TextBox =
CType(Me.frm.ActiveControl, System.Windows.Forms.TextBox)
                            If txt_ctl.Text.Length > 0 Then

cnt.Show_msg_async(cnt.Return_char(txt_ctl.Text.Substring(txt_ctl.Selecti
onStart, 1)) & " deleted")
                            Else
                                cnt.Show_msg("Text Box is Empty")
                            End If

                        Catch
```

```
                        End Try
                    End If
                ElseIf e.KeyCode = Keys.F1 Then
                    hlp.Form_help()
                ElseIf e.KeyCode = Keys.F2 Then
                    hlp.Project_help()
                ElseIf (e.Alt = True) And (e.KeyCode.ToString <> "Menu")
Then
                    hot_key(e.KeyCode.ToString)
                ElseIf (e.KeyCode = 72) And (e.Control = True) Then
                    hlp.Speak_help_message()
                ElseIf e.KeyCode = Keys.PageUp Then
                    btn.PerformClick()
                ElseIf (e.KeyCode = Keys.PageDown) Then
                    Syn.SpeakAsyncCancelAll()
                End If
            Catch ex As Exception
                cnt.Show_msg_async(ex.Message)
            End Try
        End Sub


        Sub hot_key(ByVal chr As Char)
            Try
                For Each ctl In frm.Controls
                    If (ctl.GetType.ToString <>
"System.Windows.Forms.Label") And (ctl.GetType.ToString <>
"System.Windows.Forms.GroupBox") Then
                        If ctl.AccessibleName.ToString.Length > 0 Then
                            If Char.ToUpper(chr) =
ctl.AccessibleName.Substring(ctl.AccessibleName.IndexOf("&") + 1,
1).ToString.ToUpper Then
                                If (ctl.GetType.ToString <>
"System.Windows.Forms.GroupBox") Then
                                    ctl.focus()
                                    'If ctl.GetType.ToString =
"System.Windows.Forms.Button" Then ctl.PerformClick()
                                    Exit For
                                End If
                            End If
                        End If
                    ElseIf (ctl.GetType.ToString =
"System.Windows.Forms.GroupBox") Then
                        For Each sub_ctl In ctl.Controls
                            If (sub_ctl.GetType.ToString <>
"System.Windows.Forms.Label") And (sub_ctl.GetType.ToString <>
"System.Windows.Forms.GroupBox") Then
                                If sub_ctl.AccessibleName.ToString.Length
> 0 Then
                                    If Char.ToUpper(chr) =
sub_ctl.AccessibleName.Substring(sub_ctl.AccessibleName.IndexOf("&") + 1,
1).ToString.ToUpper Then
                                        sub_ctl.focus()
                                        Exit For
                                        ' If ctl.GetType.ToString =
"System.Windows.Forms.Button" Then ctl.PerformClick()
                                    End If
                                End If
                            End If
                        Next
                    End If
```

```
                Next
            Catch ex As Exception
                cnt.Show_msg_async(ex.Message)
            End Try

        End Sub
    End Class
    ''help User by providing him the description of the current control
    Public Class help
        Dim msg As String
        Dim cnt As Show_content
        Private frm As Form
        Public Sub New(ByVal myform As Form)
            frm = myform
            cnt = New Show_content(frm)
        End Sub
        ' thisfunction display a help message to the user according to
which control the cursor i
        Public Sub Speak_help_message()
            msg = IIf(frm.ActiveControl.AccessibleDescription <> "",
frm.ActiveControl.AccessibleDescription, msgs.get_message_byID(9))
            cnt.Show_msg_async(msg + " " +
frm.ActiveControl.GetType.ToString.Substring(frm.ActiveControl.GetType.To
String.LastIndexOf(".") + 1))
            If
frm.ActiveControl.GetType.ToString.Substring(frm.ActiveControl.GetType.To
String.LastIndexOf(".") + 1).ToUpper <> "BUTTON" Then
                cnt.Show_msg_async(frm.ActiveControl.Text)
                Dim itm_cnt As Int16 = 0
                If
frm.ActiveControl.GetType.ToString.Substring(frm.ActiveControl.GetType.To
String.LastIndexOf(".") + 1).ToUpper = "COMBOBOX" Then
                    itm_cnt = CType(frm.ActiveControl,
ComboBox).Items.Count
                    cnt.Show_msg_async(msgs.get_message_byID(11) &
itm_cnt)
                    For i = 0 To itm_cnt - 1
                        cnt.Show_msg_async(msgs.get_message_byID(12) +
i.ToString + " ," + CType(frm.ActiveControl, ComboBox).Items(i))
                    Next
                End If
            End If
        End Sub
        '' form help
        Public Sub Form_help()
            msg = IIf(frm.AccessibleDescription <> "",
frm.AccessibleDescription, msgs.get_message_byID(9))
            cnt.Show_msg_async(msg)
        End Sub
        '' form help
        Public Sub Speech_Commands_help()
            msg = IIf(frm.AccessibleDescription <> "",
frm.AccessibleDescription, msgs.get_message_byID(9))
            cnt.Show_msg_async(msg)
        End Sub
        '' form help
        Public Sub Shortcuts_help()
            msg = IIf(frm.AccessibleDescription <> "",
frm.AccessibleDescription, msgs.get_message_byID(9))
            cnt.Show_msg_async(msg)
        End Sub
```

```
        Public Sub Project_help()
            msg = IIf(frm.Tag = "", "", "CURRENT PROJECT:" & " (" &
frm.Tag & ")")
            cnt.Show_msg_async(msg)
        End Sub
    End Class
    ''recognize class used to recognize user speech
    Public Class recognize
        Private gbuilder2 As GrammarBuilder = New GrammarBuilder
        Private gbuilder1 As GrammarBuilder = New GrammarBuilder
        Private gbuilder3 As GrammarBuilder = New GrammarBuilder
        Private speechgrammar2 As Grammar
        Private speechgrammar1 As Grammar
        Private speechgrammar3 As Grammar
        Private frm As Form
        Private cnt As Show_content
        Private hlp As help
        Dim msg As String
        '' recognizer used to recognize the user speech
        Private rec As SpeechRecognizer = New SpeechRecognizer
        '' synthesier used to say words

        Public Sub New(ByVal myform As Form)
            frm = myform
            hlp = New help(frm)
            cnt = New Show_content(frm)
            createGrammer()
        End Sub
        Public Sub createGrammer()
            Try
                'enable recognizer
                rec.Enabled = True ''''''''''''

                'grammer # 1
                gbuilder2.Append(New Choices("Enable", "Disable"))
                gbuilder2.Append(New Choices("Pronounicng", "Speech
Command", "Speech Dictation", "Automatic Help"))
                speechgrammar2 = New Grammar(gbuilder2)
                rec.LoadGrammar(speechgrammar2)

                'grammer # 2
                gbuilder1.Append(New Choices("Help", "Execute",
"Previous", "Next", "Read Text", "Spell Text", "Read First Word", "Read
Last Word", "Enlarge", "Reduce", "Enlarge Maximum", "Reduce Minimum",
"Home", "End", "Cancel"))
                speechgrammar1 = New Grammar(gbuilder1)
                rec.LoadGrammar(speechgrammar1)

                'grammer # 3
                gbuilder3.Append(New Choices("Shortcuts Help", "Project
Help", "Form Help", "Speech Commands Help"))
                speechgrammar3 = New Grammar(gbuilder3)
                rec.LoadGrammar(speechgrammar3)

                rec.PauseRecognizerOnRecognition = True

                '' add event handler
                AddHandler rec.SpeechRecognized, AddressOf
spRecognizer_SpeechRecognized
            Catch
                If frm.Name = "XMLBB_Interface" Then
```

```
                    cnt.Show_msg_async("Audio Devices are not installed ,
can't use speech recognition to access specific control")
                End If
            End Try
        End Sub

        Sub spRecognizer_SpeechRecognized(ByVal sender As Object, ByVal e
As SpeechRecognizedEventArgs)
            Try
                'enabling_commands class to enable or disable commands
                Dim cmds As enabling_commands = New
enabling_commands(frm)
                ' user speech
                msg = e.Result.Text

                If (msg = "Enable Pronounicng") Then
                    cmds.Enable_Cmd(1, 2)

                ElseIf (msg = "Disable Pronounicng") Then
                    cmds.Disable_Cmd(1, 3)

                ElseIf (msg = "Help") Then
                    hlp.Speak_help_message()

                ElseIf (msg = "Previous") Then
                    cnt.Show_msg_async("Previous")
                    frm.ActiveControl.ForeColor = Color.Black
                    frm.SelectNextControl(frm.ActiveControl, False,
False, True, True)

                ElseIf (msg = "Next") Then
                    cnt.Show_msg_async("Next")
                    frm.ActiveControl.ForeColor = Color.Black
                    frm.SelectNextControl(frm.ActiveControl, True, False,
True, True)

                ElseIf (msg = "Read Text") Then
                    cnt.Show_msg_async("Read Text")
                    ' speak active control text
                    cnt.Speak_control_text()

                ElseIf (msg = "Spell Text") Then
                    cnt.Show_msg_async("Spell Text")
                    ' spell active control text
                    cnt.Spell_control_text()

                ElseIf (msg = "Read Last Word") Then
                    cnt.Show_msg_async("Last Word")
                    'speak last word of active control text
                    cnt.Speak_last_word()

                ElseIf (msg = "Read First Word") Then
                    cnt.Show_msg_async("First Word")
                    'speak first word of active control text
                    cnt.Speak_First_word()

                ElseIf (msg = "Form Help") Then
                    cnt.Show_msg_async("Form Help")
                    'speak first word of active control text
                    hlp.Form_help()
```

```
                ElseIf (msg = "Project Help") Then
                    cnt.Show_msg_async("Project Help")
                    'speak first word of active control text
                    hlp.Project_help()

                ElseIf (msg = "Enlarge") Then
                    cnt.Show_msg_async("Enlarge")
                    en.enlarge()
                ElseIf (msg = "Reduce") Then
                    cnt.Show_msg_async("Reduce")
                    en.Reduce()
                ElseIf (msg = "Enlarge Maximum") Then
                    cnt.Show_msg_async("Enlarge Maximum")
                    en.Max_enlarge()

                ElseIf (msg = "Reduce Minimum") Then
                    cnt.Show_msg_async("Reduce Minimum")
                    en.Min_Reduce()

                ElseIf (msg = "Cancel") Then
                    cnt.Show_msg_async("Cancel")
                Else
                    ' can't  recognize user message
                    cnt.Show_msg_async(msgs.get_message_byID(6))
                End If
            Catch ex As Exception
                cnt.Show_msg_async(ex.Message)
            End Try
        End Sub
    End Class
    Public Class enabling_commands
        ' reference to shortcuts xml document
        Dim shrt As shortcuts = New shortcuts
        ' Dim cnt As Show_content = New Show_content
        Dim msg As String
        Dim frm As Form
        Private cnt As Show_content
        '' recognizer used to recognize the user speech
        Private rec As SpeechRecognizer = New SpeechRecognizer
        Public Sub New(ByVal myform As Form)
            frm = myform
            cnt = New Show_content(frm)
        End Sub
        '' convert the command status into enabled
        Public Sub Enable_Cmd(ByVal shrt_id As Int32, ByVal msg_id As
Int32)
            Dim enabled As Boolean =
shrt.check_shortcut_enabled_byID(shrt_id)
            If Not enabled Then
shrt.change_shortcut_enabled_status(shrt_id) '' enable pronouncing
            msg = msgs.get_message_byID(msg_id)
            cnt.Show_msg_async(msg)
        End Sub
        '' convert the command status into disabled
        Public Sub Disable_Cmd(ByVal shrt_id As Int32, ByVal msg_id As
Int32)
            Dim enabled As Boolean =
shrt.check_shortcut_enabled_byID(shrt_id)
            If enabled Then shrt.change_shortcut_enabled_status(shrt_id)
'' disable pronouncing
            msg = msgs.get_message_byID(msg_id)
```

```
            cnt.Show_msg_async(msg)
        End Sub
        '' reverse the command status
        Public Sub Change_status_cmd(ByVal shrt_id As Int32, ByVal
msg_id1 As Int32, ByVal msg_id2 As Int32)
            Dim enabled As Boolean =
shrt.check_shortcut_enabled_byID(shrt_id)
            shrt.change_shortcut_enabled_status(shrt_id)
            If enabled Then
                msg = msgs.get_message_byID(msg_id2)
            Else
                msg = msgs.get_message_byID(msg_id1)
            End If
            cnt.Show_msg_async(msg)
        End Sub
    End Class
    '' thisclass define the methods that can be used to enlarge form size
    Public Class enlarge
        Dim do_enlarge As Boolean = True
        Dim do_Reduce As Boolean = False
        Shared enlarged_value As Int16
        Dim cnt As Show_content
        Private frm As Form
        Public Sub New(ByVal myform As Form)
            enlarged_value = 1
            frm = myform
            cnt = New Show_content(frm)
        End Sub
        ' thisfunction enlarge the form to the user
        Public Sub enlarge()
            If do_enlarge Then ' allow maximum enlarge the font size of
the control to be 200
                enlarged_value = enlarged_value + 1
                do_Reduce = True
                Dim factor As System.Drawing.SizeF = New
System.Drawing.SizeF(2, 2)
                frm.Scale(factor)
                Dim ctl As Control
                Dim fnt As Font
                For Each ctl In frm.Controls
                    fnt = ctl.Font
                    ctl.Font = New Font(ctl.Font.Name, ctl.Font.Size * 2)
                    If enlarged_value = 5 Then do_enlarge = False
                Next
            Else
                cnt.Show_msg_async(msgs.get_message_byID(4))
            End If
        End Sub
        ' enlarge form size to maximum (4 doubling)
        Public Sub Max_enlarge()
            If enlarged_value < 5 Then ' allow maximum enlarge the font
size of the control to be 200
                do_Reduce = True
                do_enlarge = False
                Dim factor As System.Drawing.SizeF = New
System.Drawing.SizeF(10 / enlarged_value, 10 / enlarged_value)
                frm.Scale(factor)
                Dim ctl As Control
                Dim fnt As Font
                do_enlarge = False
                For Each ctl In frm.Controls
```

```
                    fnt = ctl.Font
                    ctl.Font = New Font(ctl.Font.Name, ctl.Font.Size *
(10 / enlarged_value))
                Next
                enlarged_value = 5
            Else
                cnt.Show_msg_async(msgs.get_message_byID(4))
            End If
        End Sub

        'thisfunction decrease the form size
        Public Sub Reduce()
            If do_Reduce Then ' allow minimum decrease of the control
font size of the control to be 10
                enlarged_value = enlarged_value - 1
                do_enlarge = True ' enable enlarging form size
                Dim factor As System.Drawing.SizeF = New
System.Drawing.SizeF(0.5, 0.5)
                frm.Scale(factor)
                Dim ctl As Control
                Dim fnt As Font
                For Each ctl In frm.Controls
                    fnt = ctl.Font
                    ctl.Font = New Font(ctl.Font.Name, ctl.Font.Size / 2)
                    If enlarged_value = 1 Then do_Reduce = False '
disable reducing since reached minimum size
                Next
            Else
                cnt.Show_msg_async(msgs.get_message_byID(5))
            End If

        End Sub

        'thisfunction decrease the form size to minimum allowed
        Public Sub Min_Reduce()

            If enlarged_value > 1 Then  ' if form is not in his minimum
size
                do_enlarge = True ' enable enlarging
                do_Reduce = False ' disable reducing
                ' define the factor of reducing form size
                Dim factor As System.Drawing.SizeF = New
System.Drawing.SizeF((6 - enlarged_value) / 8, (6 - enlarged_value) / 8)
                'reduce form size
                frm.Scale(factor)
                Dim ctl As Control
                Dim fnt As Font
                For Each ctl In frm.Controls
                    fnt = ctl.Font
                    'reduce the font size in each control
                    ctl.Font = New Font(ctl.Font.Name, ctl.Font.Size * (6
- enlarged_value) / 8)
                Next
                ' set the form enlargiing size to minimum value (intial)
                enlarged_value = 1
            Else
                ' notify user that hisform is reduced to minimum size ,
so can't reduced more
                cnt.Show_msg_async(msgs.get_message_byID(5))
            End If
        End Sub
```

```
    End Class
End Class
```

----------End of Blind Accessibility Package----------

----------Template_Form----------

```
Imports XMLBB.Blind_Accessibility

Public Class Template_Form
    'Public Project_Name As String
    Private err As Forms_Notifcation
    Public cnt As Show_content = New Show_content(Me)
    Shared last_gotFocus As String = ""
    Private loaded As Boolean = False
    Private modified As Boolean = False

    Private Sub Template_Form_Disposed(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Disposed
        Syn.SpeakAsyncCancelAll()
        If Me.Name = "XMLBB_Interface" Then
            Beep()
            Beep()
            cnt.Show_msg("Closing XMLBB")
        End If

    End Sub

    Private Sub Template_Form_GotFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.GotFocus
        If Not loaded Then
            cnt.Show_msg_async(Me.Title.Text)
        End If
        loaded = False
    End Sub

    Private Sub Template_Form_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Load
        Try
            XMLBB.Blind_Accessibility.Syn.SpeakAsyncCancelAll()
            Beep()
            If (DesignMode) Then
                Return
            Else
                msg.Text = "Template"

                err = New Forms_Notifcation

                Dim done As String =
forms_access.get_form_accessibility_details(Me, Me.GetType)

                loaded = True
                For Each ctl In Me.Controls
                    Dim fnt As Font = New Font("times new roman", 20,
FontStyle.Bold)
                    Me.BackColor = Color.White
                    ctl.ForeColor = Color.Black
                    ctl.font = fnt
```

```
                    If (ctl.GetType.ToString <>
"System.Windows.Forms.Label") And (ctl.GetType.ToString <>
"System.Windows.Forms.GroupBox") Then
                        If (ctl.GetType.ToString <>
"System.Windows.Forms.Button") Then
                            ctl.BackColor = Color.White
                        End If
                        AddHandler CType(ctl, Control).Enter, AddressOf
ctl_got_focus
                        AddHandler CType(ctl, Control).Leave, AddressOf
ctl_lost_focus
                        If (ctl.GetType.ToString =
"System.Windows.Forms.ComboBox") Then
                            AddHandler CType(ctl,
ComboBox).SelectedIndexChanged, AddressOf Ctl_Selectd_Index_Changed
                        End If
                        If (ctl.GetType.ToString =
"System.Windows.Forms.CheckBox") Then
                            AddHandler CType(ctl,
CheckBox).CheckedChanged, AddressOf Ctl_Selectd_Checked_Changed
                        End If
                        If ctl.tag <> "" Then
                            AddHandler CType(ctl, Control).Validated,
AddressOf ctl_Validated
                            AddHandler CType(ctl, Control).Validating,
AddressOf ctl_Validating
                        End If
                    End If
                    For Each sub_ctl In ctl.controls
                        Me.BackColor = Color.White
                        sub_ctl.ForeColor = Color.Black
                        sub_ctl.font = fnt
                        If (sub_ctl.GetType.ToString <>
"System.Windows.Forms.Label") And (sub_ctl.GetType.ToString <>
"System.Windows.Forms.GroupBox") Then
                            If (sub_ctl.GetType.ToString <>
"System.Windows.Forms.Button") Then
                                sub_ctl.BackColor = Color.White
                            End If
                            AddHandler CType(sub_ctl, Control).Enter,
AddressOf ctl_got_focus
                            AddHandler CType(sub_ctl, Control).Leave,
AddressOf ctl_lost_focus
                            If sub_ctl.GetType.ToString =
"System.Windows.Forms.ComboBox" Then
                                AddHandler CType(sub_ctl,
ComboBox).SelectedIndexChanged, AddressOf Ctl_Selectd_Index_Changed
                            End If
                            If (sub_ctl.GetType.ToString =
"System.Windows.Forms.CheckBox") Then
                                AddHandler CType(sub_ctl,
CheckBox).CheckedChanged, AddressOf Ctl_Selectd_Checked_Changed
                            End If
                            If sub_ctl.tag <> "" Then
                                AddHandler CType(sub_ctl,
Control).Validated, AddressOf ctl_Validated
                                AddHandler CType(sub_ctl,
Control).Validating, AddressOf ctl_Validating
                            End If
                        End If
                    Next
```

```
                Next

                Me.msg.TextAlign = ContentAlignment.MiddleCenter
                Me.Title.Text = Me.Text
                cnt.Show_msg_async(Me.Text)
            End If
        Catch ex As Exception
            msg.Text = ex.Message
        End Try
    End Sub


    Private Sub ctl_got_focus()
        Try
            Me.ActiveControl.ForeColor = Color.BlueViolet
            Dim st As String = Me.ActiveControl.AccessibleName
            If st.Length > 0 Then
                ' instance of class show content
                Dim pos As Int16 = st.IndexOf("&")
                Dim part1 As String = st.Substring(0, pos)
                Dim part2 As String = st.Substring(pos + 1)
                cnt.Show_msg_async(part1 + part2 + " " +
Me.ActiveControl.GetType.ToString.Substring(Me.ActiveControl.GetType.ToSt
ring.LastIndexOf(".") + 1))
                If (Me.ActiveControl.Text <> "") And
(Me.ActiveControl.GetType.ToString.Substring(Me.ActiveControl.GetType.ToS
tring.LastIndexOf(".") + 1).ToUpper <> "BUTTON") Then
cnt.Show_msg_async("Current Value " + Me.ActiveControl.Text)
                End If
            End If
            last_gotFocus = Me.ActiveControl.Name
            If Me.ActiveControl.GetType.ToString =
"System.Windows.Forms.TextBox" Then
                Dim txt_ctl As TextBox = CType(Me.ActiveControl, TextBox)
                txt_ctl.SelectionStart = CType(Me.ActiveControl,
TextBox).Text.Length
                txt_ctl.SelectionLength = 0
            End If

        Catch ex As Exception
            msg.Text = ex.Message
        End Try
    End Sub
    Private Sub ctl_lost_focus(ByVal sender As Object, ByVal e As
EventArgs)
        Try
            If (Me.ActiveControl.Name <> "Okay_Btn") Then
sender.CausesValidation = False
            Else
                sender.CausesValidation = True
            End If

            Syn.SpeakAsyncCancelAll()
            CType(sender, Control).ForeColor = Color.Black
            Me.ActiveControl.ForeColor = Color.Black
        Catch ex As Exception
            msg.Text = ex.Message
        End Try
    End Sub
```

```
    Private Sub ctl_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        If (Not (sender.GetType.ToString =
"System.Windows.Forms.Button")) And (Not (sender.GetType.ToString =
"System.Windows.Forms.Label")) Then
            modified = True
        End If
    End Sub

    Friend Function get_form_status() As Boolean
        Return modified
    End Function

    Public Sub ctl_Validating(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs)
        Try

        Dim ctl_tag As String = sender.tag.ToString
        If (sender.Text = "") And (ctl_tag.Contains("Required")) Then
            e.Cancel = True
            Dim err_msg As String =
err.get_control_error(sender.AccessibleDescription, "Required")
            Me.GeneralErrorProvider.SetError(sender, err_msg)
            cnt.Show_msg(err_msg)
        ElseIf (ctl_tag.Contains("Length")) Then
            Dim min_len_pos As Int16 = ctl_tag.IndexOf("Length:") +
"Length:".Length
            Dim max_len_pos As Int16 = ctl_tag.IndexOf(":", min_len_pos)
            Dim min_len As Int16 = ctl_tag.Substring(min_len_pos,
max_len_pos - min_len_pos)
            Dim max_len As Int16 = ctl_tag.Substring(max_len_pos + 1,
ctl_tag.IndexOf(",", max_len_pos) - max_len_pos)
            'determine whether the value length is within the range
            If (sender.text.length < min_len) Or (sender.text.length >
max_len) Then
                e.Cancel = True
                Dim err_msg As String =
err.get_control_error(sender.AccessibleDescription, "Length", min_len,
max_len)
                Me.GeneralErrorProvider.SetError(sender, err_msg)
                cnt.Show_msg(err_msg)
            End If
        ElseIf (ctl_tag.Contains("Range")) Then
            Dim min_len_pos As Int16 = ctl_tag.IndexOf("Range:") +
"Range:".Length
            Dim max_len_pos As Int16 = ctl_tag.IndexOf(":", min_len_pos)
            Dim min_val As Int16 = ctl_tag.Substring(min_len_pos,
max_len_pos - min_len_pos)
            Dim max_val As Int16 = ctl_tag.Substring(max_len_pos + 1,
ctl_tag.IndexOf(",", max_len_pos) - max_len_pos)
            'determine whether the value is within the range
            If (sender.text < min_val) Or (sender.text > max_val) Then
                e.Cancel = True
                Dim err_msg As String =
err.get_control_error(sender.AccessibleDescription, "Range", min_val,
max_val)
                Me.GeneralErrorProvider.SetError(sender, err_msg)
                cnt.Show_msg(err_msg)
            End If
        ElseIf (ctl_tag.Contains("Integer")) Then
```

```
            Dim Start_pos As Int16 = ctl_tag.IndexOf("Integer:") +
"Integer:".Length
            Dim End_pos As Int16 = ctl_tag.IndexOf(",", Start_pos)
            Dim err_id As Int16 = ctl_tag.Substring(Start_pos, End_pos -
Start_pos)
            If err_id = 17 Then
                Try
                    If Not (IsNumeric(IIf(sender.text = "", 0,
sender.text)) And (sender.text.ToString.IndexOf(".") = -1) And
(IIf(sender.text = "", 0, sender.text) >= -1)) Then
                        e.Cancel = True
                        Dim err_msg As String =
err.get_control_error(sender.text, err_id)
                        Me.GeneralErrorProvider.SetError(sender, err_msg)
                        cnt.Show_msg(err_msg)
                    End If
                Catch ex As Exception
                    e.Cancel = True
                    Dim err_msg As String =
err.get_control_error(sender.text, err_id)
                    Me.GeneralErrorProvider.SetError(sender, err_msg)
                    cnt.Show_msg(err_msg)
                End Try
            ElseIf err_id = 19 Then
                Try
                    If Not (IsNumeric(IIf(sender.text = "", 0,
sender.text)) And (sender.text.ToString.IndexOf(".") = -1) And
(IIf(sender.text = "", 0, sender.text) >= 0)) Then
                        e.Cancel = True
                        Dim err_msg As String =
err.get_control_error(sender.text, err_id)
                        Me.GeneralErrorProvider.SetError(sender, err_msg)
                        cnt.Show_msg(err_msg)
                    End If
                Catch ex As Exception
                    e.Cancel = True
                    Dim err_msg As String =
err.get_control_error(sender.text, err_id)
                    Me.GeneralErrorProvider.SetError(sender, err_msg)
                    cnt.Show_msg(err_msg)
                End Try
            End If
        ElseIf (ctl_tag.Contains("Custom")) Then
            Dim Start_pos As Int16 = ctl_tag.IndexOf("Custom:") +
"Custom:".Length
            Dim End_pos As Int16 = ctl_tag.IndexOf(",", Start_pos)
            Dim err_id As Int16 = ctl_tag.Substring(Start_pos, End_pos -
Start_pos)

            ' custom error # 5
            If err_id = 5 Then '' incorrect Schema name
                ' Determine whether the schema name have.xsd extension
                If
sender.text.ToString.Substring(sender.text.ToString.Length - 4).ToUpper
<> ".XSD" Then
                    e.Cancel = True
                    Dim err_msg As String =
err.get_control_error(sender.text, err_id)
                    Me.GeneralErrorProvider.SetError(sender, err_msg)
                    cnt.Show_msg(err_msg)
                End If
```

```
            End If
        End If
        Catch ex As Exception
            msg.Text = ex.Message
        End Try
    End Sub

    Private Sub ctl_Validated(ByVal sender As Object, ByVal e As
System.EventArgs)
        Try
            Me.GeneralErrorProvider.SetError(sender, "")
        Catch ex As Exception
            msg.Text = ex.Message
        End Try
    End Sub

    Private Sub Template_Form_Shown(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Shown
        Me.Title.Text = Me.Text
        If Not loaded Then
            cnt.Show_msg_async(Me.Title.Text)
        End If
        loaded = False
        For Each ctl In Me.Controls
            AddHandler CType(ctl, Control).TextChanged, AddressOf
ctl_TextChanged
            If ctl.GetType.ToString = "System.Windows.Forms.GroupBox"
Then
                For Each sub_ctl In ctl.controls
                    AddHandler CType(sub_ctl, Control).TextChanged,
AddressOf ctl_TextChanged
                Next
            End If
        Next
    End Sub

    Private Sub Template_Form_LostFocus(ByVal sender As Object, ByVal e
As System.EventArgs) Handles Me.LostFocus
        For Each ctl In Me.Controls
            ctl.CausesValidation = False
        Next
    End Sub


    Private Sub Ctl_Selectd_Index_Changed(ByVal sender As Object, ByVal e
As System.EventArgs)
        Try
            If Not Me.ActiveControl Is Nothing Then
                If sender Is ActiveControl Then
cnt.Show_msg_async("Selected Value" + " " + sender.SelectedItem.ToString)
            End If
        Catch ex As Exception
            cnt.Show_msg_async(ex.Message)
        End Try

    End Sub

    Private Sub Ctl_Selectd_Checked_Changed(ByVal sender As Object, ByVal
e As System.EventArgs)
        Try
            If Not Me.ActiveControl Is Nothing Then
```

```
                If sender Is ActiveControl Then
cnt.Show_msg_async(sender.text + IIf(sender.Checked, " is True", " is
False"))
            End If
        Catch ex As Exception
            cnt.Show_msg_async(ex.Message)
        End Try

    End Sub


End Class
```

----------End of Template_Form----------

---------- Form_Access Class----------

```
Imports System.Linq
Imports System.Configuration
Public Class forms_access
    Shared DAL_Path As String
    Shared XMLDOCFILEPATH As String
    Shared doc As XDocument = New XDocument
    Shared frm As Form
    Shared app As New AppSettingsReader()

    Shared Sub add_controls_access(ByVal frm_det As XElement, ByVal
frm_ctl As Control, ByVal ctl_name As String)
        If (frm_ctl.GetType.ToString <> "System.Windows.Forms.Label") And
(frm_ctl.GetType.ToString <> "System.Windows.Forms.GroupBox") Then
            Dim ctl_det = (From m In
frm_det.Elements("controls").Elements _
                Where m.Attribute("name").Value = ctl_name _
            Select New With { _
            .C_Name = m.Attribute("name").Value, _
            .C_Desc = m.Element("AccessibleDescription").Value, _
            .C_AccName = m.Element("AccessibleName").Value, _
            .C_Text = m.Element("Text").Value}).FirstOrDefault
            frm_ctl.Text = ctl_det.C_Text.ToString.TrimEnd().TrimStart
            frm_ctl.AccessibleDescription =
ctl_det.C_Desc.TrimEnd.TrimStart
            frm_ctl.AccessibleName = ctl_det.C_AccName.TrimEnd.TrimStart
        Else
            '' if label or group
            Dim ctl_det = (From m In
frm_det.Elements("controls").Elements _
                Where m.Attribute("name").Value = ctl_name _
            Select New With { _
            .C_Name = m.Attribute("name").Value, _
            .C_Text = m.Element("Text").Value}).FirstOrDefault
            '' any label other than message label
            If (frm_ctl.Name <> "msg") And (frm_ctl.Name <> "Title") Then
frm_ctl.Text = ctl_det.C_Text.ToString.TrimEnd().TrimStart
        End If
    End Sub

    Shared Function get_form_accessibility_details(ByVal XMLBB_frm As
Form, ByVal frm_type As Type) As String
        Try
            frm = XMLBB_frm
```

```
            XMLDOCFILEPATH = ".\DAL\Forms_access.xml"

            doc = XDocument.Load(XMLDOCFILEPATH)

            ' return form accessibility details
            Dim f_res = (From m In doc.Descendants.Elements("form") _
           Where (m.Attribute("name").Value = frm.Name) _
                    Select New With { _
                  .desc = m.Attribute("AccessibleDescription").Value, _
                  .text = m.Attribute("Text").Value, _
                  .AccName =
m.Attribute("AccessibleName").Value}).FirstOrDefault
            frm.AccessibleDescription = f_res.desc
            frm.AccessibleName = f_res.AccName
            frm.Text = f_res.text
            'return controls accessibilty details
            Dim ctl_name As String
            Dim frm_det As XElement = (From m In
doc.Descendants.Elements("form") _
                     Where m.Attribute("name").Value = frm.Name).First
            If frm_det.HasElements Then

                For Each frm_ctl In frm.Controls
                    ctl_name = frm_ctl.name
                    add_controls_access(frm_det, frm_ctl, ctl_name)
                    For Each sub_ctl In frm_ctl.controls
                        add_controls_access(frm_det, sub_ctl,
sub_ctl.name)
                    Next
                Next
            End If
            frm.Controls("Title").Text = frm.Text
            Return "True"
        Catch ex As Exception
            Return ex.Message
        End Try

    End Function

End Class
```

----------End of Form_Access Class----------

----------Form_Notification Class----------

```
Imports System.Linq
Imports System.Configuration

Public Class Forms_Notifcation
    Shared app As New AppSettingsReader()
    Shared DAL_Path As String
    Shared XMLDOCFILEPATH As String
    Shared doc As XDocument = New XDocument
    Public Function get_control_error(ByVal ctl_acc_name As String, ByVal
error_type As String) As String
        Try
            Dim error_msg As String = ""
```

```
            DAL_Path = app.GetValue("NotifyData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\Forms_Notifcation.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            ' return form accessibility details
            Dim f_res = (From m In doc.Descendants.Elements("Error") _
           Where (m.Attribute("Type").Value.ToString.ToUpper =
error_type.ToUpper) _
                    Select New With { _
                 .desc = m.Value}).FirstOrDefault

            error_msg = f_res.desc.Replace("F-1", ctl_acc_name)

            Return error_msg
        Catch ex As Exception
            Return ex.Message
        End Try
    End Function

    Public Function get_control_error(ByVal ctl_acc_name As String, ByVal
error_type As String, ByVal Val1 As Int16, ByVal val2 As Int16) As String
        Try
            Dim error_msg As String = ""
            DAL_Path = app.GetValue("NotifyData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\Forms_Notifcation.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            ' return form accessibility details
            Dim f_res = (From m In doc.Descendants.Elements("Error") _
           Where (m.Attribute("Type").Value.ToString.ToUpper =
error_type.ToUpper) _
                    Select New With { _
                 .desc = m.Value}).FirstOrDefault

            error_msg = f_res.desc.Replace("F-1", ctl_acc_name)
            error_msg = error_msg.Replace("V-1", Val1)
            error_msg = error_msg.Replace("V-2", val2)

            Return error_msg
        Catch ex As Exception
            Return ex.Message
        End Try
    End Function
    Public Function get_control_error(ByVal ctl_Text As String, ByVal
error_id As Integer) As String
        Try
            Dim error_msg As String = ""
            DAL_Path = app.GetValue("NotifyData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\Forms_Notifcation.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            ' return form accessibility details
            Dim f_res = (From m In doc.Descendants.Elements("Error") _
           Where (m.Attribute("Id").Value = error_id) _
                    Select New With { _
                 .desc = m.Value}).FirstOrDefault
            error_msg = f_res.desc.Replace("F-1", ctl_Text)
            Return error_msg
        Catch ex As Exception
            Return ex.Message
        End Try
```

```
    End Function
    Public Function get_Message(ByVal Message_id As Integer) As String
        Try
            Dim message As String = ""
            DAL_Path = app.GetValue("NotifyData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\Forms_Notifcation.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            ' return form accessibility details
            Dim f_res = (From m In doc.Descendants.Elements("Message") _
           Where (m.Attribute("Id").Value = Message_id) _
                     Select New With { _
                   .desc = m.Value}).FirstOrDefault
            message = f_res.desc

            Return message
        Catch ex As Exception
            Return ex.Message
        End Try
    End Function
    Public Function get_Message(ByVal ctl_Text As String, ByVal
Message_id As Integer) As String
        Try
            Dim message As String = ""
            DAL_Path = app.GetValue("NotifyData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\Forms_Notifcation.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            ' return form accessibility details
            Dim f_res = (From m In doc.Descendants.Elements("Message") _
           Where (m.Attribute("Id").Value = Message_id) _
                     Select New With { _
                   .desc = m.Value}).FirstOrDefault
            message = f_res.desc.Replace("F-1", ctl_Text)
            Return message
        Catch ex As Exception
            Return ex.Message
        End Try
    End Function

End Class
```

----------End of Form_Notification Class----------

----------XMLBB_Pack----------

```
Imports System.IO
Imports System.Linq
Imports System.Configuration
Class xmlbb_pack
    Shared app As New AppSettingsReader()
    Shared DAL_Path As String
    Shared XMLDOCFILEPATH As String
    Shared doc As XDocument = New XDocument
    Dim note As Forms_Notifcation = New Forms_Notifcation
    Dim Complex_attributes As List(Of Project_Object_attributes)

    '' thismethod create the main and sub directories of the new project
```

```
    Public Function create_Project_Directory(ByVal Prj_dir As String,
ByVal Sch_sub_dir As String, ByVal Frm_sub_dir As String, ByVal
Rep_sub_dir As String)
        ''project directory
        Try
            Dim Main_Dir As DirectoryInfo = New DirectoryInfo(Prj_dir)
            Main_Dir.Create()

            ' Form Sub directory.
            Dim Schema_Sub_Dir As DirectoryInfo = New
DirectoryInfo(Prj_dir & "\" & Sch_sub_dir)
            Schema_Sub_Dir.Create()

            ' Form Sub directory.
            Dim Form_Sub_Dir As DirectoryInfo = New DirectoryInfo(Prj_dir
& "\" & Frm_sub_dir)
            Form_Sub_Dir.Create()

            ' Form Sub directory.
            Dim Report_Sub_Dir As DirectoryInfo = New
DirectoryInfo(Prj_dir & "\" & Rep_sub_dir)
            Report_Sub_Dir.Create()
            Return True
        Catch ex As Exception
            Return False
        End Try
    End Function
    Public Function Delete_Project_Directory(ByVal Prj_dir As String) As
Boolean
        Dim Main_Dir As DirectoryInfo = New DirectoryInfo(Prj_dir)
        Main_Dir.Delete(True)
        Return True
    End Function

    '' thissub insert details of any new projectt into user_project
document
    Public Sub add_proj_details(ByVal Prj_dir As String, ByVal namspc As
String, ByVal Sch_sub_dir As String, ByVal Frm_sub_dir As String, ByVal
Rep_sub_dir As String)
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)
        Dim project = doc.<Projects>(0)
        Dim elm = <Project></Project>
        elm.Add(New XAttribute("Name", Prj_dir))
        Dim det = <Details></Details>
        det.Add(New XAttribute("Namespace", namspc))
        det.Add(New XAttribute("Schema", Sch_sub_dir))
        det.Add(New XAttribute("Forms", Frm_sub_dir))
        det.Add(New XAttribute("Reports", Rep_sub_dir))
        elm.Add(det)
        elm.Add(<Schemas></Schemas>)
        elm.Add(<Forms></Forms>)
        project.AddFirst(elm)
        Console.WriteLine(project)
        doc.Save(XMLDOCFILEPATH)
    End Sub
    '' used to get the details of the project (namespace, schema ,report
& forms foloders)
```

```
    Public Sub get_project_details(ByVal Project_Name As String, ByRef
namspce As String, ByRef schema_dir As String, ByRef Forms_dir As String,
ByRef Reports_dir As String)
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)

        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                  Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First
        If prj.HasElements Then
            Dim Prj_det = (From m In prj.Elements("Details") _
                    Select New With { _
                    .P_Namespace = m.Attribute("Namespace").Value, _
                    .P_Schema = m.Attribute("Schema").Value, _
                    .P_Forms = m.Attribute("Forms").Value, _
                    .P_Reports = m.Attribute("Reports").Value}).First
            namspce = Prj_det.P_Namespace.TrimEnd.TrimStart
            schema_dir = Prj_det.P_Schema.TrimEnd.TrimStart
            Forms_dir = Prj_det.P_Forms.TrimEnd.TrimStart
            Reports_dir = Prj_det.P_Reports.TrimEnd.TrimStart
        End If

    End Sub
    ' used to check if the project have already a schema created
    Public Function Project_Has_Schema(ByVal Project_Name As String) As
Boolean
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)
            Dim has_schema As Boolean = False
            Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                      Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First
            If prj.HasElements Then
                Dim Prj_Sch = (From m In prj.Elements("Schemas"))
                If Prj_Sch.Elements.Count > 0 Then
                    has_schema = True
                End If
            End If
            Return has_schema
        Catch
            Return False
        End Try
    End Function
    Public Function Schema_Has_objects(ByVal project_name As String,
ByVal schema_name As String) As Boolean
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)
        Dim has_objects As Boolean = False
        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                  Where m.Attribute("Name").Value.ToUpper =
project_name.ToUpper).First
        If prj.HasElements Then
            Dim Prj_Sch = (From m In
prj.Elements("Schemas").Elements("Schema") _
```

```
                    Where m.Attribute("Name").Value.ToUpper =
schema_name.ToUpper).First

            If Prj_Sch.Elements.Count > 0 Then
                has_objects = True
            End If
        End If
        Return has_objects
    End Function
    '' used to check if schema with same name already exist
    Public Function Duplicate_Schema(ByVal Project_Name As String, ByVal
Schema_Name As String) As Boolean
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)


            Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).FirstOrDefault
            If prj.HasElements Then
                Dim sch As XElement = (From m In
prj.Elements("Schemas").Elements("Schema") _
                              Where m.Attribute("Name").Value.ToUpper =
Schema_Name.ToUpper).FirstOrDefault
                If Not sch Is Nothing Then
                    Return True
                Else
                    Return False
                End If
            End If
        Catch
            Return False
        End Try
    End Function

    '' used to check if project with same name and path already exist
    Public Function Duplicate_Project(ByVal Project_Name As String) As
Boolean
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)


            Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).FirstOrDefault

            If Not prj Is Nothing Then
                Return True
            Else
                Return False
            End If

        Catch
            Return False
        End Try
```

```
    End Function


    '' used to check if schema object with same name already exist
    Public Function Duplicate_Schema_Object(ByVal Project_Name As String,
ByVal Schema_Name As String, ByVal Object_Name As String) As Boolean
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)


            Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).FirstOrDefault
            If prj.HasElements Then
                Dim sch As XElement = (From m In
prj.Elements("Schemas").Elements("Schema") _
                           Where m.Attribute("Name").Value.ToUpper =
Schema_Name.ToUpper).FirstOrDefault

                Dim obj As XElement = sch.Element(Object_Name)

                If Not obj Is Nothing Then
                    Return True
                Else
                    Return False
                End If
            End If
        Catch
            Return False
        End Try
    End Function

    '' insert details of schema
    Public Class Project_Object_attributes
        Public O_Desc As String
        Public O_Value As String
    End Class

    Public Sub Add_Schema_Object(ByVal Project_Name As String, ByVal
Schema_Name As String, ByVal Object_Name As String, ByVal
schema_Attribute_list As List(Of Project_Object_attributes))
        ' Try
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)

        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First

        Dim sch As XElement = (From m In
prj.Element("Schemas").Elements("Schema") _
             Where m.Attribute("Name").Value = Schema_Name).First

        If prj.HasElements Then
            Dim elm As XElement = New XElement(Object_Name)
            For Each Obj In schema_Attribute_list
```

```
                elm.Add(New XAttribute(Obj.O_Desc, Obj.O_Value))
            Next
            sch.AddFirst(elm)
        End If

        Console.WriteLine(doc)
        doc.Save(XMLDOCFILEPATH)
    End Sub
    '' modify schema element
    Public Sub Mod_Schema_Object(ByVal Project_Name As String, ByVal
Schema_Name As String, ByVal Object_Name As String, ByVal
schema_Attribute_list As List(Of Project_Object_attributes))
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                           Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First

            Dim sch As XElement = (From m In
prj.Element("Schemas").Elements("Schema") _
                 Where m.Attribute("Name").Value = Schema_Name).First

            If sch.HasElements Then
                Dim obj As XElement = (From m In sch.Elements _
                Where m.Name.ToString.ToUpper =
Object_Name.ToUpper).First
                Dim found As Boolean = False

                For Each mod_att In schema_Attribute_list
                    obj.SetAttributeValue(mod_att.O_Desc, "")
                    found = False
                    For Each att In obj.Attributes
                        If att.Name = mod_att.O_Desc Then
                            found = True
                            If mod_att.O_Value <> "" Then
obj.SetAttributeValue(mod_att.O_Desc, mod_att.O_Value)
                        End If
                        If found Then Exit For
                    Next
                Next

            End If

            Console.WriteLine(doc)
            doc.Save(XMLDOCFILEPATH)
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub

    Public Sub Add_Schema_Name(ByVal Project_Name As String, ByVal
Schema_Name As String)
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)
```

```
        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                      Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First
        If prj.HasElements Then
            Dim elm = <Schema></Schema>
            elm.Add(New XAttribute("Name", Schema_Name))
            prj.<Schemas>(0).AddFirst(elm)
            Console.WriteLine(doc)
            doc.Save(XMLDOCFILEPATH)
        End If

    End Sub
    '' used to return the list of avaliable projects
    Public Function Get_Project_List() As List(Of String)

        Dim prj_list As List(Of String) = New List(Of String)
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            Dim prj = (From m In doc.Descendants.Elements("Project"))

            For Each itm In doc.Descendants.Elements("Project")
                prj_list.Add(itm.Attribute("Name").Value)
            Next

            Return prj_list

        Catch ex As Exception
            prj_list.Add("Empty List" & ex.Message)
            Return prj_list
        End Try

    End Function

    '' used to return the list of project Schemas
    Public Function Get_Schema_List(ByVal Project_Name As String) As
List(Of String)

        Dim Sch_list As List(Of String) = New List(Of String)

        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            Dim prj = (From m In doc.Descendants.Elements("Project") _
                      Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First

            If prj.HasElements Then
                Dim Sch = (From m In prj.Elements("Schemas").Elements)
                For Each itm In Sch
                    Sch_list.Add(itm.Attribute("Name").Value)
                Next
            End If

            Return Sch_list
```

```
        Catch ex As Exception
            Sch_list.Add("Empty List" & ex.Message)
            Return Sch_list
        End Try

    End Function

    '' used to return the list of Schema objects
    Public Function Get_Schema_Object_List(ByVal Project_Name As String,
ByVal schema_Name As String, ByVal object_name As String) As List(Of
String)

        Dim Obj_list As List(Of String) = New List(Of String)

        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            Dim prj = (From m In doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First

            If prj.HasElements Then

                Dim Sch = (From m In prj.Elements("Schemas").Elements _
                    Where m.Attribute("Name").Value.ToUpper =
schema_Name.ToUpper).First

                Dim Obj = (From m In Sch.Elements)

                For Each itm In Obj
                    If object_name <> itm.Name.ToString Then
Obj_list.Add(itm.Name.ToString)
                Next
            End If

            Return Obj_list

        Catch ex As Exception
            Obj_list.Add("Empty List" & ex.Message)
            Return Obj_list
        End Try

    End Function


        '' used to check if schema used by any form
    Public Function Is_Schema_Used(ByVal Project_Name As String, ByVal
Schema_Name As String) As Boolean
        Try
            Return False
        Catch
            Return False
        End Try
    End Function

    '' thismethod can be used to delete ptoject schema
    Public Sub Delete_Project_Schema(ByVal Project_Name As String, ByVal
Schema_Name As String)
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
```

```
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)

        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First
        If prj.HasElements Then
            Dim sch As XElement = (From m In
prj.Elements("Schemas").Elements _
                      Where m.Attribute("Name").Value.ToUpper =
Schema_Name.ToUpper).First
            sch.Remove()
            Console.WriteLine(doc)
            doc.Save(XMLDOCFILEPATH)
        End If
    End Sub

    '' this method can be used to delete schema object
    Public Sub Delete_Schema_objects(ByVal Project_Name As String, ByVal
Schema_Name As String, ByVal Object_Name As String)
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)

        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First
        If prj.HasElements Then
            Dim sch As XElement = (From m In
prj.Elements("Schemas").Elements _
                      Where m.Attribute("Name").Value.ToUpper =
Schema_Name.ToUpper).First

            Dim Obj As XElement = (From m In sch.Elements _
                      Where m.Name.ToString.ToUpper =
Object_Name.ToUpper).First

            Obj.Remove()
            Console.WriteLine(doc)
            doc.Save(XMLDOCFILEPATH)
        End If
    End Sub

    '' this method can be used to delete ptoject
    Public Sub Delete_User_Project(ByVal Project_Name As String)
        DAL_Path = app.GetValue("ProjectsData", GetType(String))
        XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
        doc = XDocument.Load(XMLDOCFILEPATH)

        Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First
        If Not (prj Is Nothing) Then
            prj.Remove()
            Console.WriteLine(doc)
            doc.Save(XMLDOCFILEPATH)
        End If
    End Sub
```

```
    '' used to return  both simple and complex type objects
    Public Function return_schema_datatypes(ByVal Project_Name As String,
ByVal Schema_Name As String) As List(Of String)
        Dim type_list As List(Of String) = New List(Of String)
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)


            Dim prj As XElement = (From m In
doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).FirstOrDefault
            If prj.HasElements Then
                Dim sch As XElement = (From m In
prj.Elements("Schemas").Elements("Schema") _
                           Where m.Attribute("Name").Value.ToUpper =
Schema_Name.ToUpper).FirstOrDefault

                If sch.HasElements Then
                    Dim obj_lst = (From m In sch.Elements _
                           Where m.Attribute("Object_Type").Value.ToUpper
= "SIMPLE" Or m.Attribute("Object_Type").Value.ToUpper = "COMPLEX")

                    If obj_lst.Count > 0 Then
                        For Each el In obj_lst
                            type_list.Add(el.Name.ToString & ":" &
el.Attribute("Object_Type").Value)
                        Next
                    End If
                End If

            End If

            Return type_list
        Catch
            Return type_list
        End Try
    End Function
    Public Function check_valid(ByVal val As String, ByVal data_type As
String) As Boolean
        Try
            Select Case data_type
                Case "Decimal"
                    Dim dec_val As Decimal = CDec(val)
                Case "Integer"
                    'Dim Int_val As Integer = CInt(val)
                    If IsNumeric(val) Then
                        If val.ToString.IndexOf(".") = -1 Then
                            Return True
                        End If
                    End If
                    Return False
                Case "Boolean"
                    If IsNumeric(val) Then
                        Return False
                    Else
                        Dim Bol_val As Boolean = CBool(val)
                    End If
                Case "Date"
```

```
                    Dim Dat_val As Date = CDate(val)
                Case "Time"
                    Dim Tim_val As DateTime = CDate(val)
                Case Else
            End Select

            Return True
        Catch
            Return False
        End Try
    End Function
    Public Sub Select_schema_object(ByVal object_type As String, ByVal
project_name As String, ByVal schema_name As String, ByVal object_name As
String)

        If object_type.ToUpper = "NA" Then '' modify mode
            Dim Schema_object As XElement = Get_Object_Type(project_name,
schema_name, object_name)
            object_type = Schema_object.Attribute("object_type").Value

            If object_type.ToUpper = "ELEMENT" Then
                Dim Mod_Elm As XMLBB.Modify_Element
                Mod_Elm = New XMLBB.Modify_Element(project_name,
schema_name, Schema_object)
                Mod_Elm.Show()
                '' modify atttribute
            ElseIf object_type.ToUpper = "ATTRIBUTE" Then
                Dim Mod_Att As XMLBB.Modify_Attribute
                Mod_Att = New XMLBB.Modify_Attribute(project_name,
schema_name, Schema_object)
                Mod_Att.Show()
                '' modify simple type
            ElseIf object_type.ToUpper = "SIMPLE" Then
                Dim Mod_Simple As XMLBB.Modify_Simple_Type
                Mod_Simple = New XMLBB.Modify_Simple_Type(project_name,
schema_name, object_name, Schema_object)
                Mod_Simple.Show()
            Else
                '' modify complex type
                Dim mod_complex As XMLBB.Modify_Complex_Type
                mod_complex = New XMLBB.Modify_Complex_Type(project_name,
schema_name, object_name, Schema_object)
                mod_complex.Show()
            End If
        ElseIf object_type.ToUpper = "ELEMENT" Then
            Dim New_Elm As XMLBB.New_Element
            New_Elm = New XMLBB.New_Element(project_name, schema_name,
object_name)
            New_Elm.Show()
            '' create new atttribute
        ElseIf object_type.ToUpper = "ATTRIBUTE" Then
            Dim New_Att As XMLBB.New_Attribute
            New_Att = New XMLBB.New_Attribute(project_name, schema_name,
object_name)
            New_Att.Show()
            '' create new simple type
        ElseIf object_type.ToUpper = "SIMPLE" Then
            Dim New_Simple As XMLBB.New_Simple_Type
            New_Simple = New XMLBB.New_Simple_Type(project_name,
schema_name, object_name)
            New_Simple.Show()
```

```
        Else
            '' create new complex type
            Dim New_Complex As XMLBB.New_Complex_Type
            New_Complex = New XMLBB.New_Complex_Type(project_name,
schema_name, object_name)
            New_Complex.Show()
        End If
    End Sub

    '' used to return the list of Schema objects
    Public Function Get_Object_Type(ByVal Project_Name As String, ByVal
schema_Name As String, ByVal object_name As String) As XElement
        Dim Sch As XElement
        Dim obj As XElement = New XElement("empty")
        Try
            DAL_Path = app.GetValue("ProjectsData", GetType(String))
            XMLDOCFILEPATH = DAL_Path + "\User_Projects.xml"
            doc = XDocument.Load(XMLDOCFILEPATH)

            Dim prj = (From m In doc.Descendants.Elements("Project") _
                       Where m.Attribute("Name").Value.ToUpper =
Project_Name.ToUpper).First

            If prj.HasElements Then

                Sch = (From m In prj.Elements("Schemas").Elements _
                    Where m.Attribute("Name").Value.ToUpper =
schema_Name.ToUpper).First

                If Sch.HasElements Then
                    obj = (From m In Sch.Elements _
                    Where m.Name.ToString.ToUpper =
object_name.ToUpper).First
                End If

            End If

            Return obj
        Catch ex As Exception
            Return obj
        End Try
    End Function
End Class
```

---------- End of XMLBB_Pack ----------